intel®

APPLICATION NOTE

AP-52

March 1979

Using Intel's Industrial Control Series in Control Applications

Peter Andersen
OEM Microcomputer Systems Applications

I. INTRODUCTION

The introduction of the single board computer as a tool for the system designer has opened the way for many varied application areas to benefit from the advantages of computer utilization. A problem still exists, however, because the available I/O configurations have been largely incompatible with the wiring and packaging techniques required in industrial environments. This problem is overcome by the utilization of the Intel® iCS™ product family. The purpose of this application note is to provide a representative approach to the implementation of a computerized solution to an industrial control system.

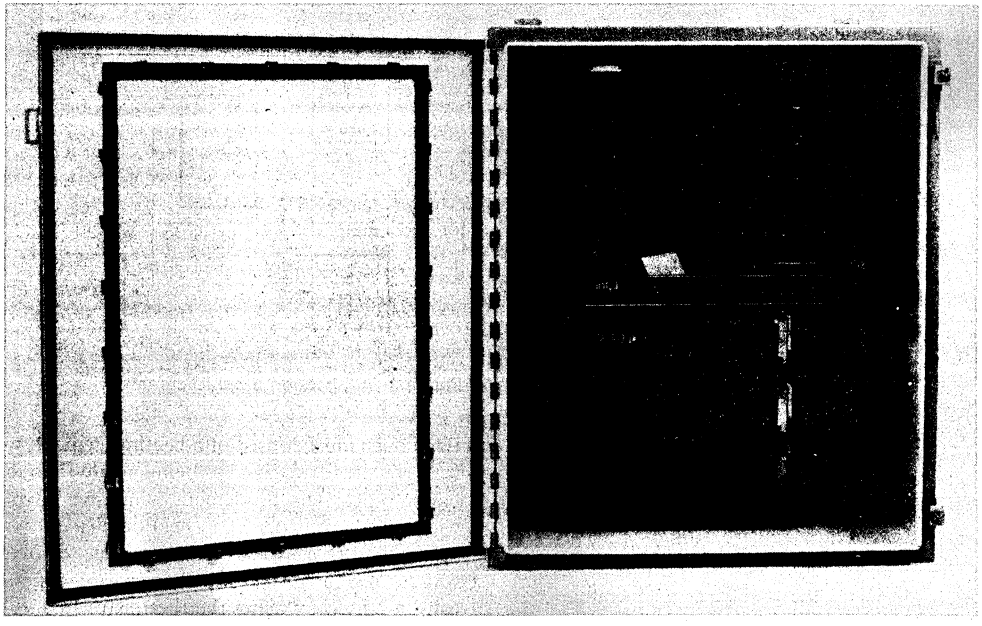
System Description

This application note will deal with a control system which will regulate the temperature in each of four ovens. Each oven will be defined as utilizing a light bulb for heating. Normal convection will be used to provide cooling. The internal temperature will be measured by means of a thermistor installed in each oven. We will assume that we will be required to implement some type of operator panel near the ovens which will allow the status of each oven to be monitored. This approach is similar to many common industrial applications which require a supervisory control station in one area and a separate operator interaction panel near the equipment being controlled. The setpoint and tolerances should be input from an external location.

With these facts about our system defined, we can begin a step by step solution to providing a computerized control system to operate the ovens. We will discuss the various equipment trade-offs and the decisions which will be used to define the hardware/software designs.

Control Algorithm

Before we can begin the design of our system, we must have a clear idea of the technique we will use to control the system. Our control system must maintain the oven temperature within a predefined and fairly narrow range of the setpoint. Let us make an assumption that the light bulb will be controlled digitally, meaning that the bulb must either be turned fully on or it must be turned fully off. The obvious control technique then becomes turning the bulb on when the temperature of the oven is below our lower limit and turning the bulb off when the temperature is above the higher limit. It seems reasonable to assume that this technique will provide a temperature in the oven which varies sinusoidally with time. This is true because even though the lamp is turned off, it will continue to generate heat for a short period of time. Likewise, when the bulb is turned on, it will not instantly be able to provide heat to raise the temperature of the



chamber. We would expect to have a system response such as is shown in Figure 1. A better method of control can be devised if we provide some type of temperature prediction into our control algorithm. Since this utilizes the rate of temperature increase or decrease, it will involve a type of derivative control system. This derivative control action will tend to dampen the temperature oscillations which might be encountered if only an instantaneous on-off control system were utilized. Figure 2 shows the response with time that we might expect with this type of control system.

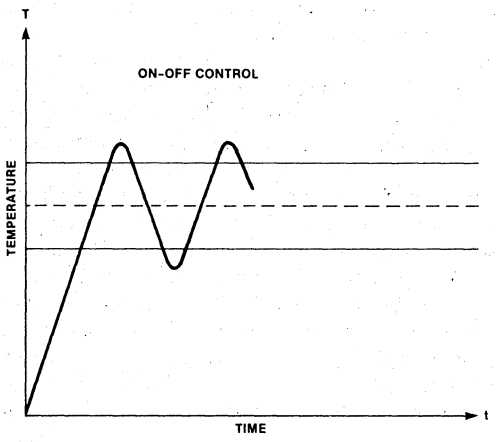


Figure 1. Maximum Effort Current Temperature

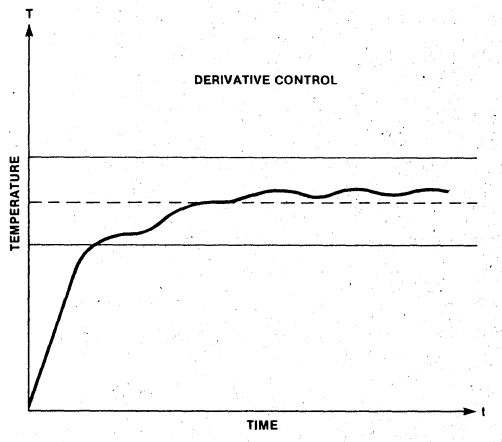


Figure 2. Maximum Effort Projected Temperature

The second approach is superior to the first because the control will provide a much smaller oscillation of the oven temperature. Other solutions are possible, such as providing a modulated output to the lamp. However, in an attempt to provide a simple model upon which to expand our system solution, we will assume that the second approach will provide us with an accurate enough control of the oven temperature.

Having made the decision as to the control technique, we can proceed with the task of determining the general system configuration. That is, we can define the physical system characteristics and the components to which we must interface the computer system. This approach is identical to that which would be used in a conventional control system design.

Basic System Configuration

Based upon the data which we have provided so far, it is possible to build a block diagram of the system's major components. The system consists of four ovens, an operator's panel, a data entry panel, and the actual control logic. A block diagram for the system is shown in Figure 3. We must now further define the elements which make up each of these blocks.

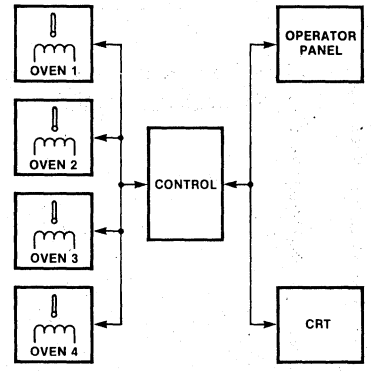


Figure 3. Application Block Diagram

Each oven must consist of a heating element, which we have already defined as being a light bulb, and a temperature sensing element which we have said will be a thermistor. Each heating element will be switched on or off by applying or removing a source of 115 VAC. The thermistor temperature can be sensed by using the thermistor in a voltage

divider circuit. We can then measure the voltage across a fixed resistor to obtain an analog signal which is proportional to the oven temperature. We will determine the required value of the fixed resistor at a later time.

The operator's panel should be designed to provide the workfloor operator with basic information as to the status of each oven. It should also allow some method by which he can inhibit the operation of any oven should it become necessary for charging or servicing the oven. We can then define the basic elements which should make up the operator's control panel. Each oven should have associated with it the following controls and indicators:

1. Oven ON/OFF Switch — This switch will allow the operator to inhibit the oven operation by turning the appropriate oven switch to OFF.
2. Oven RUNNING Indicator — This indicator will provide a visual indication that the oven is activated and that the temperature is being controlled.
3. Oven IN TOLERANCE Indicator — This indicator will turn on when the oven temperature falls within the allowable bandwidth around the setpoint for that oven.
4. Oven ALARM Indicator — This indicator is the complement of the in tolerance lamp. It will be turned on when the oven is activated and the temperature does not lie within the desired bandwidth.
5. Oven CAUTION Indicator — It may be necessary to alert the operator to a potential oven temperature control problem before it actually occurs and sets off the alarm indicator. Since we have defined our control algorithm as utilizing a type of derivative control, we can project the oven temperature ahead in time. We will turn the oven caution indicator on when we predict that the oven temperature will lie outside of the desired bandwidth in a predetermined future time period.

We have now defined the operator interface which we will utilize to control and monitor the oven processes.

At this point, we will make a decision that the interface used to input the setpoints will utilize a CRT terminal. Though the decision may seem to be completely arbitrary, we will see later that CRT terminals provide an extremely useful device for allowing an operator to communicate with the system. Once the decision has been made, we have no further requirements to consider hardware design for this terminal, as the entire operation can be handled in the software development which will be considered later.

A common technique for documenting a system is the ladder diagram. At this time, we can construct a ladder for our control system. Unlike conventional design techniques, our ladder diagram need only be concerned with the actual drive and sensing circuits since the logic required to drive the various outputs will be defined using software. This results in a considerable simplification of the design process. A ladder diagram for a typical oven is shown in Figure 4. We can defer the implementation of the control algorithm until we begin to develop the software portion of our control system. It is now possible to complete the external hardware design and to implement the system wiring package.

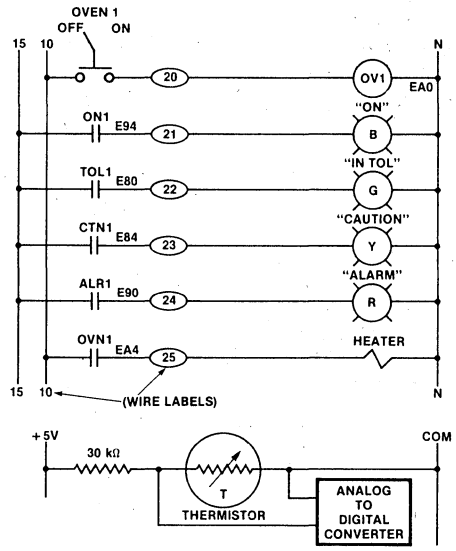


Figure 4. Ladder Diagram of One Oven

II. WIRING INTERFACES

A major pitfall in utilizing a computer for control systems has traditionally been the requirement for the design engineer to expend a considerable amount of his time in designing interfaces to connect the physical wiring to the computer system. The introduction of Intel's product line of termination panels has essentially eliminated the require-

ment of designing interfaces and allows more engineering time to be spent providing a solution to the application. Before we continue with the specific design, we should spend some time discussing the various types of termination panels available and the general characteristics of each panel.

Analog Termination Panels

The Intel® iCS 910 Analog Termination Panel has been designed to provide a simple means of terminating the analog wiring and of providing an interface to the control system input/output. All wiring is terminated utilizing pressure type screw barrier blocks. Termination blocks have been provided to allow the termination of up to 32 single-ended or 16 differential channels of analog input. For use in a differential input environment, such as we will be using, the terminator blocks provide wiring terminations compatible with shielded cable inputs in that provision has been made to accept the shield of each input signal. The shield is then carried through the on-board circuits to the analog-to-digital converter. Provision has been made on the board for the mounting of commonly used circuits for signal conditioning. The available signal conditioning circuits provide for installation of current termination resistors and the installation of a single pole low pass filter network. The basic barrier assignments for the iCS 910 termination panel are shown in Figure 5. The possible circuit networks for this panel are illustrated in Figure 6. A complete description of the analog termination panel can be found in the *iCS 910 Analog Signal Conditioning/Termination Panel Hardware Reference Manual* (manual order number 9800800A).

The functions of the analog termination panel will become more clear as we develop the actual configuration required to support our oven application. Referring to the ladder diagram (Figure 4) we see that a fixed resistor is necessary to provide the voltage divider network to sense the oven temperature. The current termination resistor (Rc) on the iCS 910 board can be used to provide a convenient mounting location for this component (refer to iCS 910 circuit schematic, Figure 6). At this point, we must make a design decision regarding the utilization of a low pass filter for our analog circuits. Since the oven temperatures are not expected to exhibit rapid fluctuations with time, the use of a low pass filter will not adversely effect the temperature

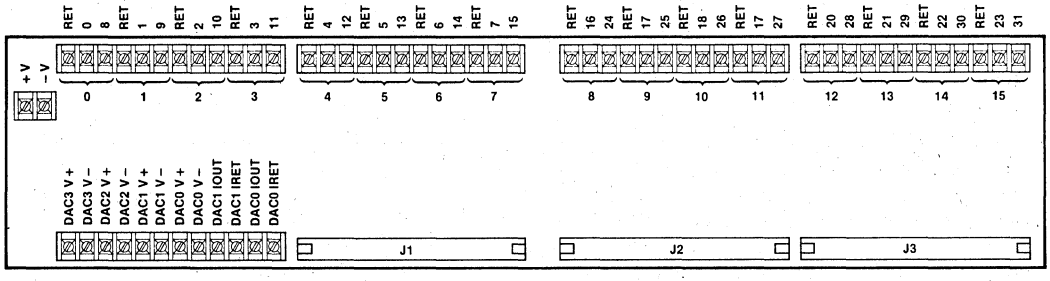


Figure 5. iCS™ 910 Analog Terminator Panel Assignments

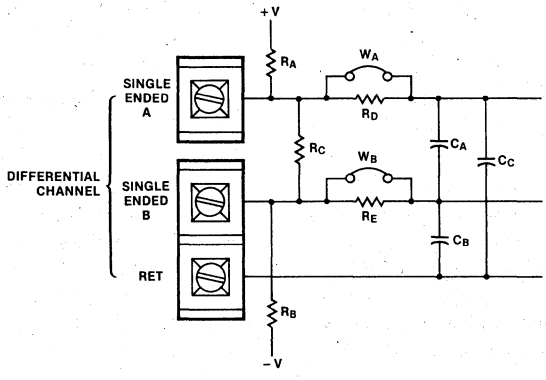


Figure 6. Typical Circuit on Analog Terminator

sensing. Indeed, the use of a low pass filter should contribute to spurious signal rejection should the analog cables pick up external noise signals. Calculations will show that the use of a filter network consisting of 11K ohms series resistors and a 2.2μF capacitor will provide the filter characteristics shown in Figure 7.

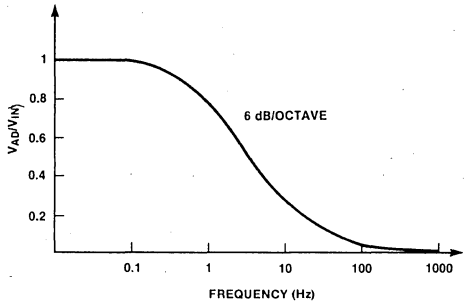


Figure 7. Single Pole Filter Characteristics

Based upon our requirements and using the circuit schematic of Figure 6, we can provide the circuit interfaces required by our ladder diagram (Figure 4) by configuring the channels of the iCS 910 terminator as shown in Figure 8. This results in a simple two-wire per oven analog interface. The terminator board is designed to connect to the various analog I/O boards by means of a standard ribbon cable which is supplied with the terminator panel. The actual selection of the appropriate analog board will be deferred until later. We will define that oven number 1 will correspond to the differential analog channel 0; oven will correspond with channel 1; oven 3 will correspond with channel 2; and oven 4 will use channel 3. This leaves 12 analog differential channels available for future expansion. The channel selection just made was a purely arbitrary choice.

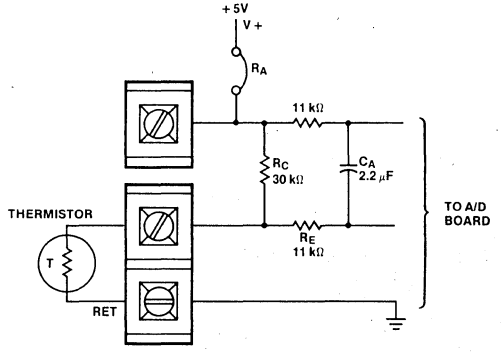


Figure 8. Analog Circuit for Oven Application

The wiring to the iCS 910 terminator panel can then be made essentially as shown in Figure 9. Clearly, the use of the terminator panel greatly simplified the connection between the control sys-

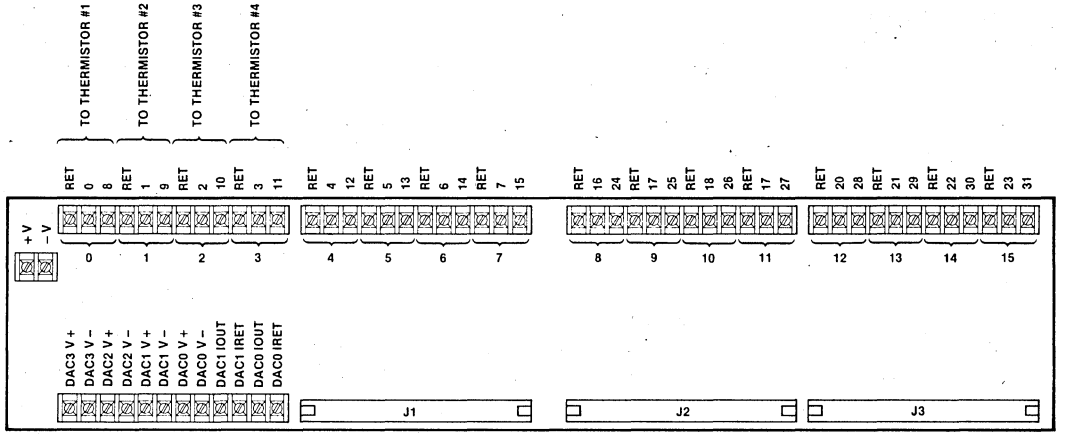


Figure 9. Analog Terminator Wiring

tem and the physical devices which are to be monitored or controlled. Figure 10 shows the placement of the components onto the board.

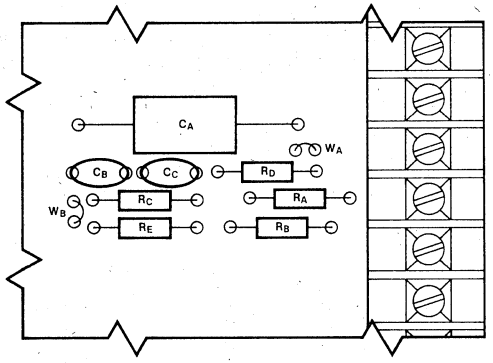


Figure 10. Analog Terminator Component Locations

Low Voltage Digital Termination Panels

Looking again at our ladder diagram for an oven control system (Figure 4), we see the need to provide a second type of interface signal. This is to provide the switching for the various indicator lamps used on the operator's control panel. Traditionally, this interface has been handled by using electromechanical relays. The coils would be driven by the low voltage control system and the relay contacts were used to drive the external indicators. Modern technology provides us with a solid state device to perform the same function, the optical isolator. We can use these devices to provide a highly reliably and low cost alternative to the relay interface. The Intel® iCS 920™ Digital Signal Conditioning/Terminator Panel provides us with a convenient vehicle for mounting the optical isolator circuits and for terminating the wiring associated with the indicator devices.

The iCS 920 panel is designed to be used by those interface circuits which incorporate operating voltages less than 50 volts and which generally use currents which are smaller than 300 mA. These limits are given only for a general guideline since a wide variety of optical isolators and drivers are available for use on the board. Some of the devices are capable of handling greater voltages or currents. A representative list of available devices and complete details of the termination panel are available in the *iCS 920 Digital Signal Conditioning/Termination Panel Hardware Reference Manual* (manual order number 9800801A).

The digital panel provides terminations for up to 24 digital channels, each of which can be configured as either an input or an output channel according to the specific application requirements. As with the analog termination panel, all wire terminations are made using pressure type barrier strips which will accept up to 16 gauge wire. The 24 digital channels correspond with those input/output channels assigned to the standard Intel I/O configurations used on the single board computers and I/O expansion boards. We will dwell more on this subject later when we define the addresses associated with each circuit which we desire to incorporate into the termination panel.

Since the digital channels can be configured into either an input or an output mode, it is wise to discuss each configuration so that a clear understanding of the board can be obtained, even though our application example will only use the output mode with this board.

Figure 11 provides a schematic of the panel when it is configured for a digital input mode. To set up a channel to operate as an input, it is necessary to add at least two jumpers to the wire-wrap jumper posts. As can be seen, pins 6 and 4 must be connected together as well as pins 3 and 5. If the board is to provide a visual LED indication of the channel status, an additional jumper should be installed between pins 1 and 2 of the jumper posts. If this is done, be certain to take into account the additional current requirements when calculating the required input resistors. Two resistor mounting locations are provided to allow installation of selected components to handle the current limit through the optical isolator (Rx) and the threshold voltage for turn-on of the device (Ry). A complete and detailed procedure for selecting these resistors based upon the input voltages is provided in the iCS 920 hardware reference manual mentioned earlier. Provision has also been made on the termination panel for the installation of a diode (CR) to protect against reverse bias application.

The components have been placed on the board arranged in groups of two channels. This eases the task of finding various components or of locating the holes for installing the required components. This layout is illustrated in Figure 12. It is important to take note of the physical placement of the optical isolator chips in the 20-pin socket. This installation location must be followed rigorously when using a channel in an input mode. Also take note that provisions are provided for mounting two sizes of resistors in location (Rx). This will accom-

modate the power dissipation requirements which will be encountered in various application situations. Referring again to Figure 12, note that the **upper half of the layout represents odd channels and the lower portion of the layout is used for even channel component mounting.**

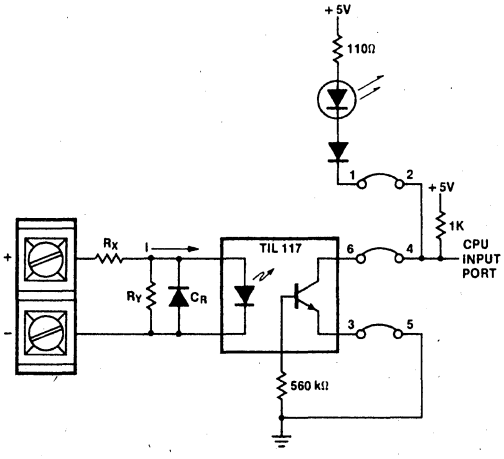


Figure 11. iCS™ 920 Digital Terminator Input Configuration

When the iCS 920 panel is used in this input mode, it corresponds to the utilization of a relay coil to sense some external contact closure. The resistors can be thought of as selecting the coil's operating voltage and the diode provides the same transient protection function as when installed on an electromechanical relay. Finally, the optical isolator output corresponds to the contacts associated with the relay coil. **As we will see later, this approach provides us with an unlimited number of contacts per relay coil.**

The oven application requires a contact for driving the indicator lamps associated with each oven. If we define the driving voltage to be 24 volts DC, we will find that the voltage and current requirements fall within the limits specified for using the iCS 920 Digital Signal Conditioning/Termination Panel. Let us examine in more detail how this can be accomplished.

We will select an industrial indicator assembly which utilizes a full voltage 24-volt lamp. Typical lamps would be type 387. This will require a drive of 40 mA at 28 volts. Our switching device must be capable of driving this load. The analogy used earlier to compare the optical isolator with a relay in an input mode holds true when we utilize the devices in an output configuration. If we examine the data sheet for the current switching characteristics of a typical optical isolator, say the TIL 113 (Appendix A), we can see that the current and voltage requirements fall well within the allowable ratings of the device. We have selected the relay contact characteristics! We need not concern ourselves with the selection of current limitation resistors (coil voltage ratings) since this circuitry is provided on the terminator panel when a circuit is

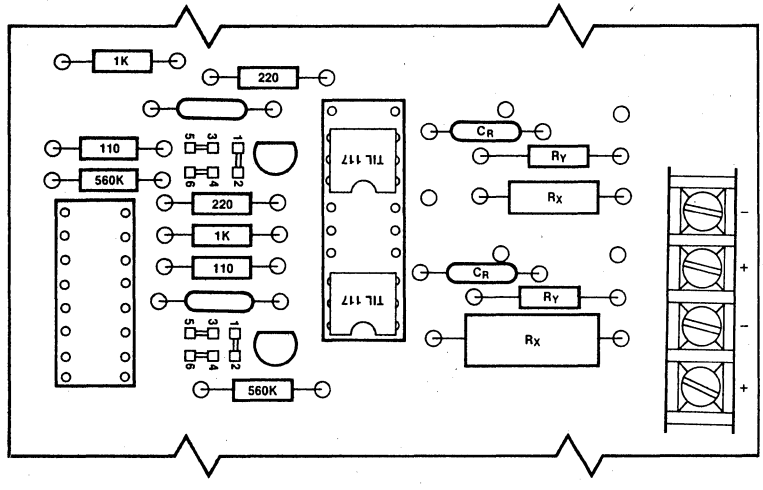


Figure 12. Digital Terminator Input Parts Layout

configured in an output mode. If we refer to Figure 13, we can see the on-board schematic for the output drive mode of operation. Two jumpers must be installed for each output channel. The first, between pins 1 and 2, is used to enable the LED channel status indicator. The second, between pins 3 and 4, actually connects the computer generated drive signal to the input of the optical isolator (analogous to connecting the relay coil to the driving line). Provision has been made on the circuit board for only one optional component in the output mode; this is the resistor (Rz). This component has the effect of increasing the response time of the switching device. Because our indicator lamps are not time critical, we will choose to omit the installation of this component.

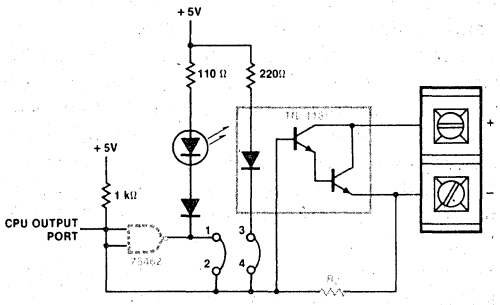


Figure 13. iCS™ 920 Digital Terminator Output Circuit

Figure 14 provides a drawing showing the location of the components on the iCS 920 panel when it is utilized as an output switch. Again note the placement of the optical isolators in the 20-pin sockets. Also note the jumper arrangement used to provide the required output circuitry.

Again referring to Figure 13, we see that an alternative to using the optical isolator for a switch exists. Provision has been made on the panel for the installation of high power buffer/driver chips such as the TI 75462. This device provides the same coil/contact characteristics as our optical isolator; however, no isolation between the input and output is provided. In certain applications, this configuration may be desirable and can be implemented by connecting jumpers 1 and 3 together, then placing a jumper block in the isolator socket location. The oven application will not use this mode because of the many advantages which isolation can provide.

Prior to actually installing the components onto the iCS 920 panel, it is necessary to assign the lamps to definite channel addresses. This involves making some additional assumptions and design configuration decisions. If we consider the total number of digital inputs and outputs which are required to handle all four ovens (including the as yet unconsidered switch and heater signals), we see that a total of 24 channels will be required. These will be broken out as shown below:

No. of Channels	Type	Function
16	DC	Oven indicator lamps
4	AC	Oven heaters
4	AC	Oven RUN switches

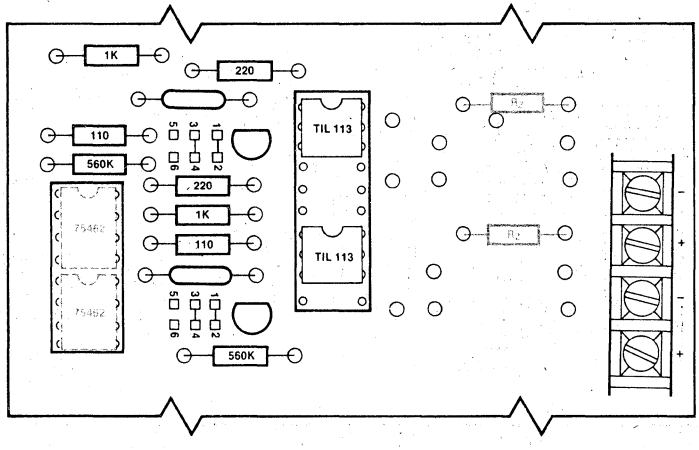


Figure 14. Digital Terminator Output Configuration

We have indicated that the 16 indicator lamps can be handled using the iCS 920 panel. An examination of the data sheets for the various Intel single board computers and expansion boards provides us with the fact that a common characteristic of most boards is the use of at least one Intel 8255 Programmable Peripheral Interface. This provides us with at least 24 I/O lines with which to work on each single board computer. We can then assume that we will not require an I/O expansion board to implement our application. Ideally, we can handle our total requirements with one parallel interface.

The various Intel parallel ports are brought off of the computer and expansion boards using edge connectors. These edge connectors are then connected to the termination panels using a standrd ribbon cable assembly, effectively providing an extension of the I/O ports out to the termination panels. The 24 channels are grouped into three I/O ports (each consisting of 8 channels or bits) which are then called port A, port B, and port C. When connected to the iCS 920 panel, these ports and their bit assignments will be as shown in Figure 15.

At this point, we seem to be in a dilemma since we would like to use all 24 channels and we have used only 16 of them on our panel while we have utilized the edge connector of the interface. It would be desirable to have some technique to extend the other 8 channels to a high voltage terminator panel. It might be well to interrupt our channel assignments at this time to jump ahead and consider the features of the iCS product line which will enable us to accomplish our interface desires. We will then consider the interface of the high voltage signals to our control system before returning to the problem of assigning port locations to our lines.

High Voltage Digital Termination Panels

The Intel® iCS 930™ AC Signal Conditioning/ Termination Panel is designed to interface up to 16 AC signals (up to 280 volts at 3 amps) or high current DC signals (up to 50 volts at 3 amps) to the parallel ports of the Intel single board computers or I/O expansion modules. The barrier strip terminations on this panel are designed to easily handle the 14 gauge wire commonly found in applications requiring the use of the AC terminator.

Solid state relays are used to provide the interface between the computer I/O ports and the physical plant devices. These devices make the utilization of the panel a simple task once a ladder diagram of the required circuits has been drawn. As we have previously mentioned and as is clear from looking at Figure 4, we shall need to utilize eight of the available circuits, four for input and four for output. The implementation of each signal type requires only that we insert the correct type of solid state relay into the appropriate socket.

First, consider the input configuration which is required to sense the position of the oven RUN switches. Figure 16 shows the circuit schematic when used in the input mode. We can see that the output signal will turn on when the input power is applied. Like the digital termination panel, each circuit's status is indicated by means of an LED indicator installed on the board. The input circuit is protected by a socketed 3-amp fuse which may be replaced without the need to solder any components. The solid state relay used for this configuration should be a type IAC5 which is available from either Opto-22 or Motorola. Complete details of available relays and their uses on the board are available in the *iCS 930 AC Signal Conditioning/*

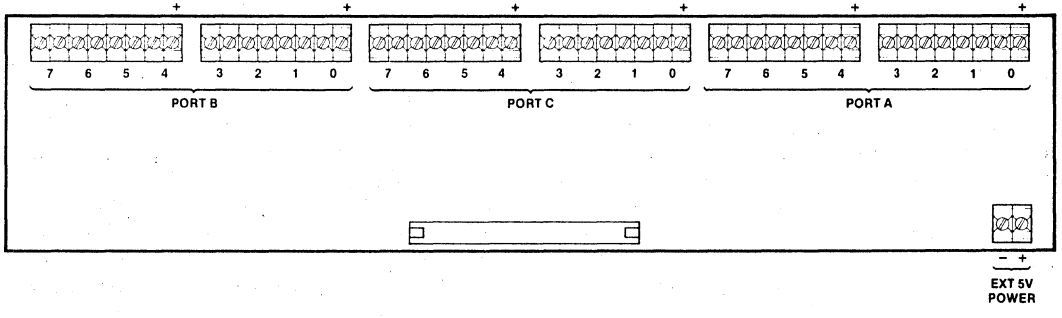


Figure 15. iCS™ 920 Digital Terminator Port Assignment

AFN-01931A

Termination Panel Hardware Reference Manual (manual order number 9800802A). Keep in mind the fact that although this application note represents the solid state relays as being actual relays and contacts, they in fact are solid state and contain no moving parts.

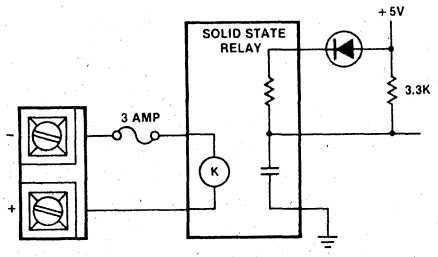


Figure 16. iCS™ 930 AC Terminator Input Circuit

The output configuration is utilized to turn the heater elements (the light bulbs) on and off. Figure 17 provides us with a schematic of the output circuitry. In this case, we will insert a solid state relay of type OAC5 which will handle up to 140 volts RMS at 3 amps. In some cases, it might be desirable to add certain components to the terminator panel when using it in the output mode. Two possible circuit configurations are possible. The first and perhaps the most common will consist of installing a MOV (metal oxide varistor) across the solid state relay contacts. This will be required when the load being driven is inductive in order to prevent the transients generated by the load from damaging the triac in the SSR (solid state relay). Since the SSRs utilize zero voltage switching and the load in our ovens is resistive rather than inductive, our application will not necessitate the installation of this device. The second possibility for additional circuitry also involves driving inductive loads. When the load is highly inductive, a possibility exists that reliable operation of the SSR may not occur because of incorrect values for the dv/dt (a complete description of this phenomenon is available in various publications available from the manufacturers of the solid state relay devices). Provision has been made for installation of an external snubber network should this be required. Again, our oven control system will not require this type of circuitry. Figure 18 is provided for reference should the reader desire to see the location of the additional components on the panel. It should be noted that the component placement does not allow the installation of the MOV and the snubber simultaneously.

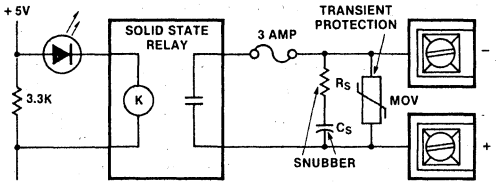


Figure 17. iCS ™930 AC Terminator Output Circuit

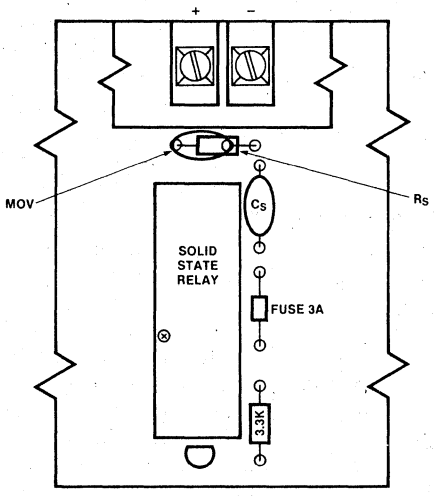


Figure 18. AC Terminator Component Locations

We can now get back to the task of assigning addresses to the various digital channels. The iCS 930 panel has three connector options for connecting it to the computer's I/O ports. The standard configuration utilizes connector J2 to attach the ribbon cable assembly. When this is done, the computer ports A and B will correspond to the 16 channels on the terminator panel (Figure 19). If we look at the termination panel, we will see that there is a provision for the user installation of two additional ribbon connector sockets onto the board. These are used in order to utilize the computer port C. If connector J3 is installed and utilized instead of J2, the channel assignments will be as shown in Figure 20. In a similar manner, connector J1 can be installed and utilized to provide connections between the computer port C and the other eight SSR positions. If we choose the 16 lines required for driving

the indicator lamps from the iCS 920 panel to be ports A and B, then it seems reasonable to assign the eight remaining lines required on the iCS 930 to port C. A feature of utilizing standard ribbon cable assemblies is the ability to easily add ribbon plug connectors to the cable. This will result in an assembly transferring ports A, B and C to the iCS 920 panel (however, port C is not used) and which continues the port C signals to the iCS 930 panel.

Individual channel assignments can now be made, grouping the inputs and outputs together in groups of four (this is done because of a requirement of the single board computers to share terminator and driver component packages in groups of four). Figure 21 provides a drawing showing the channel assignments and the physical wiring locations which will be used to connect the oven heaters and switches.

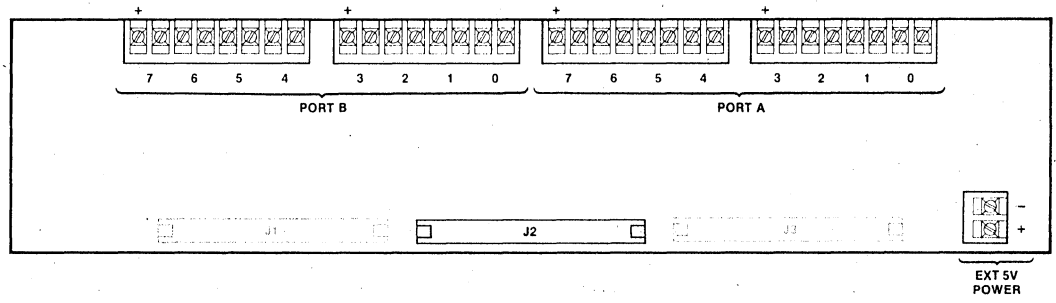


Figure 19. iCS™ 930 AC Terminator Port Assignments

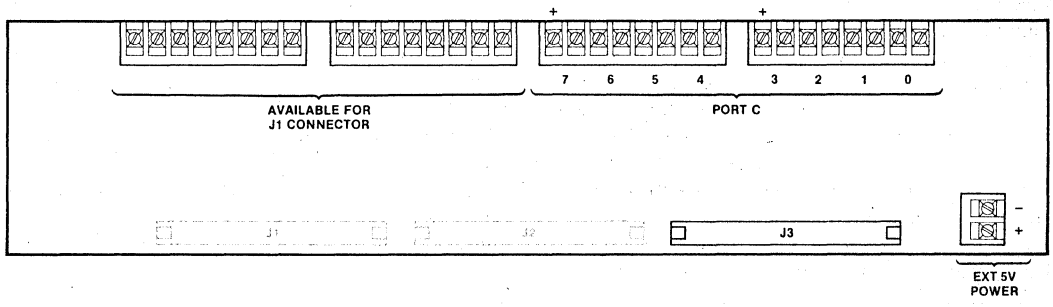


Figure 20. iCS™ 930 AC Terminator Port Assignments

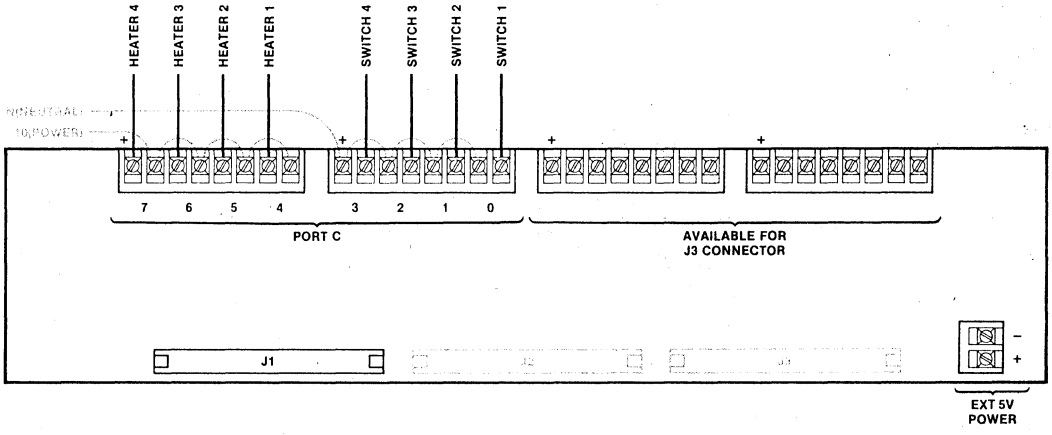


Figure 21. iCS™ 930 AC Terminator Application Configuration

Final Channel Assignments

The only task remaining before we have completed our task of assigning channel numbers and physical wire and component locations is to assign these channels on the iCS 920 digital termination panel. Since we have already determined that we will utilize ports A and B, this becomes a simple matter, requiring only an arbitrary assignment of lamp locations using these port bits. The assignments made for one oven can be seen in Figure 22. The entire ladder diagram of the system can now be completed along with port assignments for all signals used. The completed diagram can be found in Appendix B. Note how the port assignments have been shown to the side of the ladder element representing that interface device.

The method used to define a port assignments needs to be clarified since it may not be apparent why a channel of port A was given the address of E80. To begin, we have already indicated that each port consisted of eight channels or bits. We will number these bits from 0 to 7. Since it is possible to have many input/output devices connected to the computer, the possibility exists of having multiple devices which incorporate internally ports A, B, and C. The computer has been designed to support up to 256 of these ports so we have numbered them using the hexidecimal numbering system. The possible port numbers can then range from 00 to FF. It will be found that a common characteristic of most single board computers is the use of assigning the port addresses of E8, E9, and EA to the on-board 8255 parallel peripheral interface. Therefore, the first channel of port A would be defined as having an address of E80; the second channel of port B would be E91, and so forth.

III. SELECTING THE COMPUTER BOARDS

To this point we have delayed the selection of the boards which will be required to provide the computerized control system. The Intel OEM Microcomputer Systems Configuration Guide has been designed to simplify the task of selecting the required system. Our first task is to enter all known information describing our desired system into the project configuration worksheets. These worksheets can then be used to actually select a board configuration which meets our particular requirements. The effort required to accomplish the entry of data is reduced to a minimum through the use of predefined digital and analog configuration worksheets. Our requirement of having a total of 24 parallel data lines, consisting of a mix of high and low level interfaces, can be met by the 24-bit AC/DC combination. Our assignments of requirements for the terminator panels can be made and is shown in Figure 23. It can clearly be seen from the worksheets, that our required interface with the computer digital data will consist of one 24-bit wide connector (had we not used port C assignments, the use of 16-bit wide connectors would have sufficed). This means that our selected single board computer or I/O expansion board must provide at least one edge connector having 24 I/O bits on it.

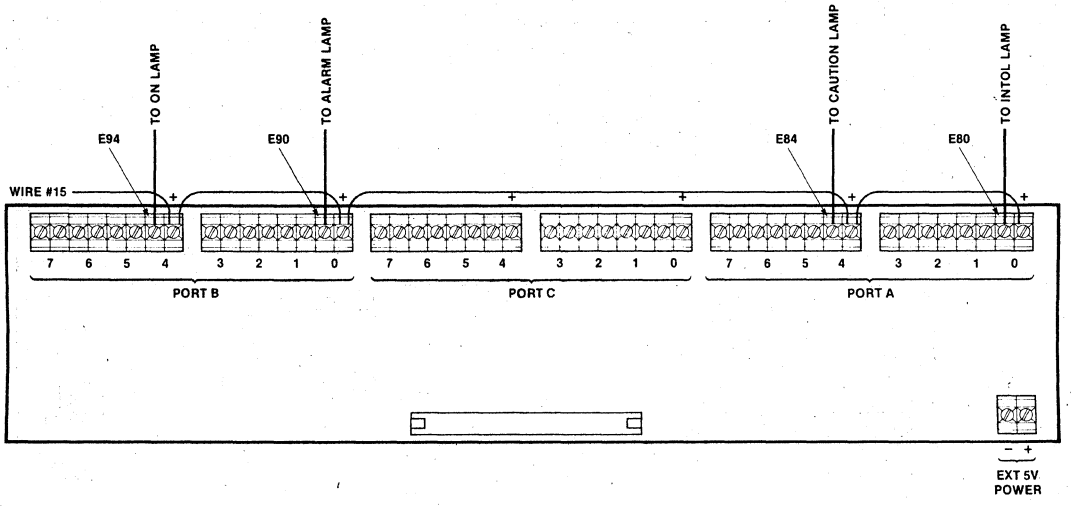


Figure 22. Digitsl Panel Application Configuration

DIGITAL CONFIGURATION WORKSHEET

PROJECT _____

This worksheet will provide the required digital interface configuration data which is required to complete the Project Configuration Worksheet.

Enter Number of Channels

Enter # of Discrete AC Outputs (115-230 VAC)	4	(A)
Enter # of Discrete AC Inputs (115-230 VAC)	4	(B)
Enter # of Discrete DC Outputs (Current > 300 MA)	0	(C)
Enter # of Discrete DC Outputs (Current < 300 MA)	16	(D)
Enter # of Discrete DC Inputs	0	(E)

Compute the Number of iCS 920™ and iCS 930™ Termination Panels

First compute the number of Parallel I/O ports (8-bits each port) required on your iSBC™ board. Round all computations up to the nearest whole integer unless instructed otherwise!

Compute # of iCS 930 Interface Output Ports ((A+C)/8)	1	(F)
Compute # of iCS 930 Interface Input Ports (B/8)	1	(G)
Compute # of iCS 930 Termination Panels ((F+G)/2)	1	(H)
Compute # of iCS 920 Interface Output Ports (D/8)	2	(J)
Compute # of iCS 920 Interface Input Ports (E/8)	0	(K)
Compute # of iCS 920 Termination Panels ((J+K)/3)	1	(L)

Optimization of Digital I/O Port Usage for Minimum I/O Configuration

Compute # of iCS 930 Output "Overflow Channels" DO NOT ROUND OFF) (A+C)/8	QUOTIENT	0	(M)
(Overflow Channels)	REMAINDER	4	(N)
Compute # of ICS 930 Input Overflow Channels (DO NOT ROUND OFF) (B/8)	QUOTIENT	0	(P)
	REMAINDER	4	(R)
Compute # of iCS 920 Output Overflow Channels (DO NOT ROUND OFF) (D/8)	QUOTIENT	2	(S)
	REMAINDER	0	(T)
Compute # of iCS 920 Input Overflow Channels (DO NOT ROUND OFF) (E/8)	QUOTIENT	0	(V)
	REMAINDER	0	(W)
Compute 8-Bit Input Ports Required (P+V)		0	(X)
Compute 8-Bit Output Ports Required (M+S)		2	(Y)
Compute 4-Bit Output Ports Required ((N+T)/4) (ROUND UP)		1	(Z)
Compute 4-Bit Input Ports Required ((R+W)/4) (ROUND UP)		1	(AA)
Compute 8-Bit Port C Requirements ((Z+AA)/2) (ROUND UP)		1	(BB)
Total I/O Parallel Ports Required (X+Y+BB)		3	(CC)
Total # of 24 Channel Parallel I/O iSBC Board Edge Connectors (CC/3) (ROUND UP TO INTEGER)		1	(DD)

Compute Power Requirements for the Termination Boards (DO NOT ROUND OFF)

Compute +5V for iCS 920 Board Outputs (.061 × D)	.976	(EE)
Compute +5V for iCS 920 Board Inputs (.023 × E)	0	(FF)
Compute +5V for iCS 930 Board Outputs ((.020 × (A+C))	.080	(GG)
Compute +5V for iCS 930 Board Inputs (.012 × B)	.048	(HH)
Compute iCS 920 Power Requirements (EE+FF)	.976	(JJ)
Compute iCS 930 Power Requirements (GG+HH)	.123	(KK)

Enter the appropriate data into the Project Configuration Worksheet as shown below:

SEE INSTRUCTION SHEET

PROJECT CONFIGURATION WORKSHEET

EQUIPMENT PARAMETERS:

YOUR REQUIREMENTS / PRODUCT	MEMORY: RAM (Bytes)	MEMORY: EPROM (Bytes)	SERIAL I/O: Serial Ports: TTY	SERIAL I/O: Serial Ports: RS232C	PARALLEL INPUT/OUTPUT: Parallel Lines: I/O	PARALLEL INPUT/OUTPUT: Parallel Lines: OUT	PARALLEL INPUT/OUTPUT: Interrupts: I/O	PARALLEL INPUT/OUTPUT: Interrupts: 24	ANALOG INPUT/OUTPUT: Analog: DI	ANALOG INPUT/OUTPUT: Analog: DI	ANALOG INPUT/OUTPUT: Analog: LOW	ANALOG INPUT/OUTPUT: Analog: HI	ANALOG INPUT/OUTPUT: Analog: HI	ANALOG INPUT/OUTPUT: Analog: OUT	POWER REQUIREMENTS: +5V	POWER REQUIREMENTS: +12V	POWER REQUIREMENTS: -5V	POWER REQUIREMENTS: -12V	COST: Unit	COST: Qty	COST: Qty	COST: Qty
(L) ea iCS-920					(F)	(D)									(JJ)							
(H) ea iCS-930					(B)	(AAC)		(BB)							(KK)							
TOTAL																						
INTEL SOLUTION / SLOT #1																						

Figure 23. Digital Configuration Worksheet

The required power requirements of the termination panels can be calculated using the data provided in the digital configuration worksheet. The information regarding the necessary connectors and the power requirements should then be transferred to the project configuration worksheet (Figure 24).

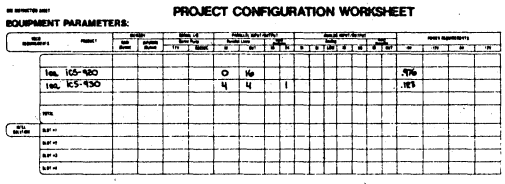


Figure 24.

A similar technique is used to configure the analog signals using the standard analog configuration worksheet as shown in Figure 25. It can be seen that our application will require a single cable connection to a differential input edge connector of an analog input board. The power requirements can be calculated from the current requirements to drive the thermistors and the sensing resistors. The data is entered into the appropriate columns of the configuration tables and then transferred to the project configuration worksheet.

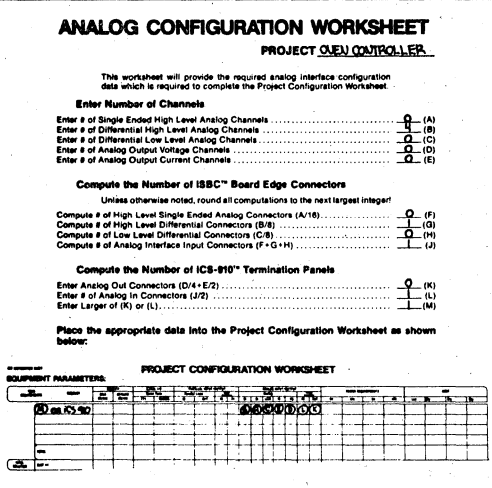


Figure 25.

The only remaining physical element of our control system which we have not defined is the CRT terminal which will be used for setpoint entry and modification. Communications with a terminal requires that we provide a serial RS232C port in our control system. This port requirement is entered onto the worksheet and the system requirements are totaled as shown in Figure 26.

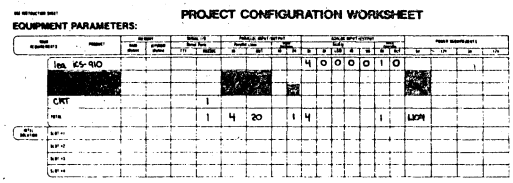


Figure 26.

We must now choose the Intel iSBC boards which will provide a solution to our system requirements. This is done by referencing the summary of key iSBC configuration parameters to find boards which provide the necessary characteristics. Our first task is to choose a single board computer which meets as many of our needs as is practical, while providing performance characteristics adequate to our needs.

Our first requirement for having support for a single RS232C serial communications channel can be seen to be met by a variety of possible boards. Among the possible boards meeting this requirement are:

iSBC 86/12™ iSBC 80/10A™
iSBC 80/20™ iSBC 80/20-4™
iSBC 80/30™

We must look further before a final choice can be made. Again, it can be seen that all candidates also meet the requirement of providing a minimum of one 24-bit wide digital I/O connector. Our decision must be based upon parameters which are not necessarily related to the input or output capabilities. Even though we have not yet developed our software package for our control system, we can safely make some assumptions regarding the completed software package and thus define additional requirements which will enable us to select our desired computer board. The software task will be considerably simplified if we write our programs in a high level language and if we use available drivers for our input and output where they are available. As we will see, the utilization of PL/M and RMX/80™ real-time executive and drivers will make this programming task much less demanding of our time. The trade-off is that these software tools take larger amounts of memory than if we were to write our entire application program in assembly language. Let us make an initial estimate that our system will require about 8K of EPROM and in the neighborhood of 2K of RAM.

Entering this data on the configuration worksheet (Figure 27) enables us to narrow our choice by eliminating the iSBC 80/10A since it does not have sufficient RAM on board.

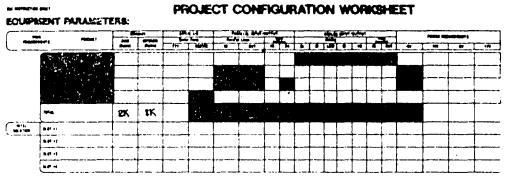


Figure 27.

Since our application is not likely to require extensive math handling capabilities or high speed capabilities, we probably do not need the power found in the iSBC 86/12; so we will remove this product from consideration.

We are now faced with selecting either the iSBC 80/20 board or the 80/30 board for our processor. Each has certain advantages and disadvantages for use in our application. Let's compare these two boards, considering first the iSBC 80/20, then the iSBC 80/30.

iSBC 80/20 board advantages — Slightly lower cost, greater number of I/O lines available.

iSBC 80/30 board advantages — Faster processor, dual ported memory, able to utilize UPI modules.

If the system were to operate in a stand-alone environment and we could be certain that significant expansion would not take place, we would probably choose the iSBC 80/20 computer for our application. If we consider that the system might become a part of a much larger system by future expansions and additions, we should remember that the use of the UPI modules on the iSBC 80/30 computer provides considerable power through multiprocessing capabilities. The dual ported memory can also provide us with the ability to use more sophisticated inter-board communication protocol should the need arise. For the purposes of this application note, we will assume the system is being designed for expansion and we will select the iSBC 80/30 computer.

A good design practice is to provide an extra margin of available memory in the hardware design. Our anticipated RAM memory will use about 2K bytes. The computer will provide us with 4K bytes so we have a considerable margin. This is not true when we look at the amount of EPROM available on the board. Our 8K requirement is identical to the amount of memory available to us on the board. We should consider the use of an expansion EPROM board or the prospect of having to spend a considerable amount of time reworking our program to get it to fit if we find that we have exceeded our estimates. We will select the option of adding a memory expansion board (it can be deleted if we find that our software requirements are less than estimated).

The computer selection and the memory expansion board data can now be entered onto the configuration worksheet as shown in Figure 28. If needed, the addition of the memory expansion board will allow our EPROM requirements to grow up to 16K bytes.

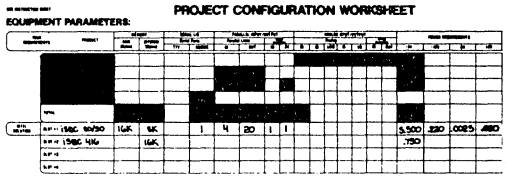


Figure 28.

The only requirement which we have not met is to assign a board to handle the analog input needs of our temperature sensing circuit. The analog voltage can be calculated and will be found to lie in the neighborhood of 4.6 volts at room temperature. This value will increase toward 5 volts as the temperature of the oven increases. Since we have no requirement for any analog output capabilities, we will choose the Intel® iSBC 711™ Analog Input Board to sense the voltage level. This board can be configured to handle a 5-volt full scale input and will provide a resolution of 12 bits. (If an oven requiring a wide range of temperatures and greater resolution were required, we would have to reconfigure our temperature sensor to provide a wider voltage spread over operating temperatures. For purposes of simplicity and clarity we will assume that our temperature resolution is adequate.)

The configuration worksheet can be filled in to reflect the selection of the analog converter and the total power requirements for the system can be computed as has been done in Figure 29. We now need to select a chassis and power supply in order to complete the application hardware design phase.

The Industrial Chassis

Before the boards can be operated together to form a control system, a means of allowing communica-

SEE INSTRUCTION SHEET

PROJECT CONFIGURATION WORKSHEET

EQUIPMENT PARAMETERS:

YOUR REQUIREMENTS	PRODUCT	MEMORY: RAM (Bytes)	MEMORY: (E)PROM (Bytes)	SERIAL I/O: Serial Ports: TTY	SERIAL I/O: Serial Ports: RS232C	PARALLEL INPUT/OUTPUT: Parallel Lines: IN	PARALLEL INPUT/OUTPUT: Parallel Lines: OUT	PARALLEL INPUT/OUTPUT: Input Connectors: 16	PARALLEL INPUT/OUTPUT: Input Connectors: 24	ANALOG INPUT/OUTPUT: Analog: DI	SI	LOW	IO	VO	ANALOG INPUT/OUTPUT: Analog Connectors: IN	OUT	POWER REQUIREMENTS: -5V	-12V	5V	12V	COST: List	Qty	Qty	Qty
	Iso. iCS-80									4	0	0	0	0	1	0								
	Iso. iCS-80					0	16										.896							
	Iso. iCS-930					4	4		1								.125							
	CRT				1																			
	TOTAL	2K	8K		1	4	20		1	4						1	1.021							
INTEL SOLUTION	SLOT #1 iSBC 80/30	16K	8K		1	4	20	1	1								3.500	.880	.0025	.050				
	SLOT #2 iSBC 416		16K														.730							
	SLOT #3 iSBC 711									8						1	1.700							
	SLOT #4																							
	SLOT #5																							
	SLOT #6																							
	SLOT #7																							
	SLOT #8																							
	SLOT #9																							
	SLOT #10																							
	SLOT #11																							
	SLOT #12																							
	ROM/EPROM																.090							
	I/O DRIVERS TERMINATION																.420							
	USER SUPP HARDWARE																							
	SUBTOTAL																8.668	.220	.0025	.050				

SYSTEM INTEGRATION	SLOTS AVAILABLE	POWER AVAILABLE: -5V	-12V	5V	12V
SYSTEM					
CHASSIS iCS 80-35	4	4.0	2.0	9.00	1.00
CARD CAGE					
POWER SUPPLY					
				TOTAL SYSTEM COST	

NOTE:
DI-Differential Input
SI-Single ended Input
IO-Current Output
VO-Voltage Output

Figure 29.

tion between the boards and of distributing power among the boards must be found. This requirement is met by specifying a chassis into which the boards will be mounted. The Intel® iCS 80™ Industrial Chassis provides an environment for operating the boards which is specifically designed to operate in an industrial area.

The chassis has been designed to facilitate mounting into either a standard 19-inch RETMA cabinet or it may be rear-panel mounted into an enclosure such as may be found in applications requiring the use of a NEMA electrical enclosure. The card chassis has been mounted in such a manner as to hold the single board computers and expansion modules vertically, facilitating maximum cooling of the boards. Fans are provided to aid the normal convection cooling process. Card racks may be installed into the iCS 80 chassis to expand the card support capability to a maximum of 12 card slots in groups of four. Either an iSBC 635 or 640 power supply can be mounted into the industrial chassis to provide power up to 4 or 12 boards capability, respectively.

Our application design requires the installation of a three board solution, so we will choose the iCS 80 chassis with one iSBC 635™ power supply. We will choose to mount our control system in a standard NEMA 12 enclosure to protect the unit from the industrial environment. We should refer to the *iCS 80 Industrial System Site Planning and Installation Guide* (manual order number 9800798) for complete details for selecting appropriate enclosures and installation instructions.

The +5 volt power needed to support the various termination panels and to supply a reference voltage for the thermistors is available from a barrier strip located on the lower front of the iCS 80 chassis (Figure 30). Our wiring can be routed to this barrier strip for those circuits requiring either 5-volt DC or the system logic common. A fuse holder is provided and a fuse should be installed for system protection. We will install a 2-amp fuse into the holder (our maximum power requirement for external circuitry should be 1.22 amps according to Figure 26).

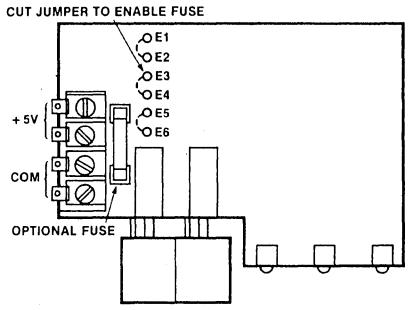


Figure 30. Industrial Chassis DC Power Strip

The remaining terms required in our ladder diagram (Appendix B) consist of a high voltage neutral and a source of switched high voltage power for the heater lamps. Both of these terms are available from the iCS 80 industrial chassis. It is desirable to utilize the same switched power for both the computer system and our external signals, so that we can provide protection to operators when one portion of the system is shut down. A common source will insure that all portions of the system are inactivated if repair is being done. The iCS 80 chassis incorporates a heavy duty industrial key-lock switch for its power switching. The outputs of this switch are available to the user at a terminal barrier strip located on a fold-out panel on the rear of the chassis assembly (refer to Figure 31). We can see that our neutral wire should be connected to terminal 5 (filtered AC low) and the wire for the AC high, wire #10 on the ladder diagram, should be connected to terminal 9. This will provide us with a switched, fused, and filtered power source for our external wiring.

As we will be installing the chassis into a NEMA enclosure, we will not want to use a standard power cord since this would involve the additional expense of installing a duplex outlet in the cabinet. The power wiring can be installed directly onto the power barrier strip by placing the AC hot wire on barrier number 1, the neutral wire onto barrier number 4, and the ground onto barrier number 3.

The hardware implementation of the system can now be considered to be complete. Before the system can function as a control for the oven temperatures, we must define the relationships between the various pieces of the oven system and we must also define the operator interface with the CRT terminal. Thus, we begin the software phase of our design.

IV. DETERMINATION OF SOFTWARE APPROACH

The task of providing the relationships between the various system components falls into the category of writing the software. Before we actually begin to develop this software, we will define certain guidelines which can be used to organize and simplify the task.

Let us consider the general environment under which our programs will operate. We find that we have essentially two choices in this area. First, we can consider the entire process as a sequential set of predefined operations in which we must perform each operation before moving to the next until finally we complete the sequence and begin again. (This is analogous to using a single stepper switch to design our control system.) Since each oven is independent of the others, we can not afford to use

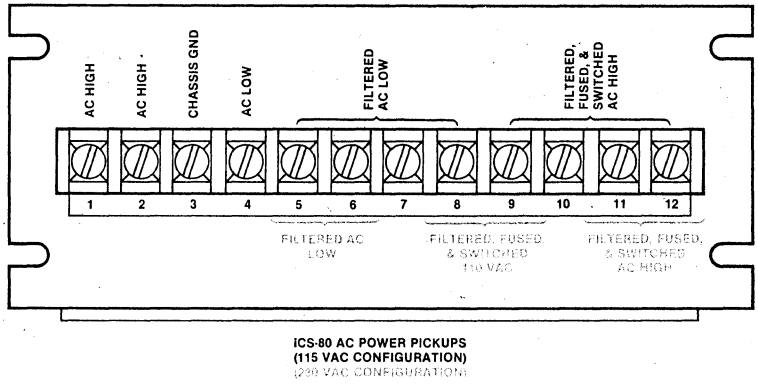


Figure 31. Industrial Chassis AC Power Strip

this approach since we could get tied up waiting for something to happen in a particular oven and would have to ignore the other ovens. The designer familiar with relay design will probably be thinking, at this point, that we should use a separate sequential operation for each oven or device to be controlled. Indeed, this is exactly what we can do with our software by using what is known as a real-time executive. This tool will allocate the computer's resources in such a manner as to provide us with the capability of having independent software programs or tasks operating at what appears to be the same time. We will make our first assumption that our software will be written using such a tool and we will specify that we will operate under Intel's RMX/80 Real-Time Multi-Tasking Executive. We will discuss more detail of this software tool as we develop our programs.

Next, we must consider the language which we will use to actually define our required operation. We have many alternatives from which to choose. Let us look at several of the alternatives in some detail.

Assembler

Assembler language is probably the most basic tool with which we can program a computer. It is considered to be the most efficient user of program memory and processor time. These features are made possible because each assembler instruction line is converted directly into a corresponding machine instruction. From a programming standpoint, assembler language is the most difficult to use since any task must be defined by subdividing that task into a multitude of smaller operations compatible with the available instructions of the computer. To use this language, we must be familiar with the architecture of each computer with which we desire to operate. The use of the language is somewhat simplified through the use of an Intel supplied assembler which converts the assembler code into machine instructions and provides listings of the operations which have been entered. A complete description of the Intel 8080/8085 Assembler Language is available in the *8080/8085 Assembly Language Programming Manual* (manual order number 9800301B).

The user should consider this programming tool when his application requires the minimum amount of memory (such as might be required for very large volume designs where memory cost is a factor) or where a highly time dependent routine must be defined. Our oven application does not fall into either of these categories, so we will choose not to use this language in our instance.

PL/M

Intel's PL/M language offers an efficient, structured, high level systems programming language. Before proceeding, let us be clear on the benefits of using a high level language. First, the use of high level languages results in reduced development time and cost. High level languages provide the ability to program in a natural algorithmic language. In addition, they eliminate the need to manage register usage or to allocate memory. Second, high level languages provide improved product reliability because programs tend to be written in structured formats and result in a minimum of extraneous branches which might cause testing problems. Finally, their use produces programs which are better documented and are easier to maintain.

On the other hand, high level languages do not optimize the code segments as well as can be done by an experienced assembly language programmer. As a result, most compilers (routines which convert the high level languages into machine executable code) use more program storage than those written by the assembly language programmer. Different languages and compilers require different amounts of memory for the same task.

PL/M-80 is probably one of the most efficient high level languages for use on microcomputers. It has been determined that PL/M-80 users can expect to use between 1.1 to slightly more than 2 times as much program memory as would be used for the same task written in assembly language. For this reason, we must place the use of this language high upon our list of possible languages in this application.

A glance at the *PL/M-80 Programming Manual* (manual order number 98-268B) indicates that the language is highly structured and seems to lend itself very well to handle logical type operations. It seems to have the greatest weakness in its math handling capabilities in that it does not support negative numbers or fractions. It is reasonable to assume that the oven application can be handled entirely with positive integer numbers so this limitation will not unduly hamper our use of this language. We will keep these features in mind when making a final decision.

FORTRAN

Intel's FORTRAN-80 provides the full subset of ANSI FORTRAN 77. In many cases FORTRAN-80 has features that exceed the specifications for both the subset and the full versions of FORTRAN 77. Most of the power of this language lies in its ability to easily handle complex mathematical expressions. Obviously, it does not have any limitations regarding fractions or sign of the numbers involved. It should be used when the application requires the use of mathematical computations. The power of the language, however, means that the use of the language will take a heavy toll of memory allocation. A complete description of the FORTRAN version supported by Intel and its use on the iSBC computers can be found in the *FORTRAN-80 Programming Manual* (order number 9800481A) and in the *ISIS-II FORTRAN-80 Compiler Operator's Manual* (order number 9800480).

It is unlikely that the magnitude of mathematical routines required to control the temperature of our ovens will be complex enough to justify the use of FORTRAN. Keep in mind that, if such a situation were encountered, it is feasible to use a combination of programming languages to create our final module.

BASIC

Certaintly the most well known high level programming language today is BASIC. It offers a quick way of applying the computational capabilities of the computer to a wide range of applications. The Intel RMX/80 BASIC-80 is an interpreter designed to operate with Intel's single board computers and contains extended disk handling capabilities. As an interpreter, it differs from other high level languages in that it results in a relatively slower operating solution to an application. It is also not possible to use BASIC to generate multiple independent tasks which can compete for computer resources.

For these reasons, we cannot consider the use of BASIC for a solution to our application.

Final Selection of Language

From the above discussion, it seems clear that our choice for the application being demonstrated is to use PL/M-80 as our programming language.

With this in mind, we can begin the task of actually generating the code which will complte our application and provide an operating control system.

V. DEFINING SOFTWARE TASKS

The software implementation can begin as soon as we have broken our control functions into independent "tasks". We can then handle each task separately as though it were the only thing which had to be done by the control system. In the event that we find that one of our tasks must communicate with or be interlocked with another, we will handle this need through the use of "exchanges". The "exchange" can be thought of as a mailbox into which messages are deposited and picked up by the various tasks. These messages convey the necessary information between the otherwise independent programs. When all tasks have been coded, we will combine them using the facilities of RMX/80.

Our oven application can be broken down into three functional areas or tasks. These are:

1. The Control Task which will be used to actually sense the oven temperature and to provide the required responses to the heaters and the indicator lamps.
2. The CRT Update Task will be used to provide a "snapshot" of the system operations to a person viewing the CRT terminal.
3. The Parameter Update Task will be used to examine and update the oven setpoints and tolerances.

The choice of these three tasks has been essentially arbitrary in nature. Certainly, other choices and groupings of functions could easily have been made. We will use these choices for our example and will proceed with our development accordingly.

We have two other supporting tasks which must be included in our system. Fortunately, these tasks are predefined and fully supported within RMX/80's libraries; thus we need not write these functions. The two supporting tasks are:

4. A Terminal Handler Task to support the actual interface to the CRT terminal. It provides echo of input characters and signals when data is ready to be read. It will output messages to the terminal and signal when all characters requested have been sent.
5. An Analog I/O Driver Task to request and handle the handshaking which is required to communicate with the analog input board. It will signal us when data has been input and is available for use by our user written tasks.

We can proceed with the implementation of each of our three tasks which we have defined. The first step with each will be to develop a flowchart which shows the required operations to implement that task. This flowchart will show any intertask communications or exchanges that may be required tunately, the voltage and temperature are not using the facilities provided by our programming language.

Oven Control Task

The sequence of operations required to perform the control task can be defined using the flowchart shown in Figure 32. Let us examine the required steps in more detail.

An arbitrary decision has been made to only sample and control the ovens once each second. This will allow some time for the system to respond once a heater output has been set. The first step in our control task is to wait for one second to elapse.

Our next subtask should be to read the status of the various oven control switches on the operator's control panel. This item could wait until a later time, but there is no harm in handling it at this time.

Next, we see a block indicating the input of data regarding the current oven temperatures. This oven temperature data will certainly be used by the task handling the snapshot display on the CRT so we must give some consideration to the validity of the data. While we are in the process of getting the data and converting it to engineering units (next step), there will be periods during which the stored temperature data does not reflect the actual oven temperature. An example might be when we are actually moving the 16 bits of the temperature since we can only move data 8 bits at a time. During this period, we would not want another task to use the data and since each task is going to operate independent of others, we must provide some type of lockout of the data while we are operating on the temperatures (an alternative would be to have each task get its own temperature from the A/D converter and convert it to engineering units, but this would seem to waste memory and computer time). We can provide this lockout by creating an exchange to communicate with other tasks. If we make a message available in this exchange when the data is valid and cause no messages to be available when the data is nonvalid, we can effectively lock out tasks from using the data when it is in the process of being updated. This is done by requiring those tasks to test for the presence of a message at the exchange before they get the temperature data. If no message is present, they must wait until one is placed into the exchange before proceeding. Just before we update the temperatures we will fetch the message from the exchange, leaving it empty while we work on the data. later we will again restore the message when the update is complete.

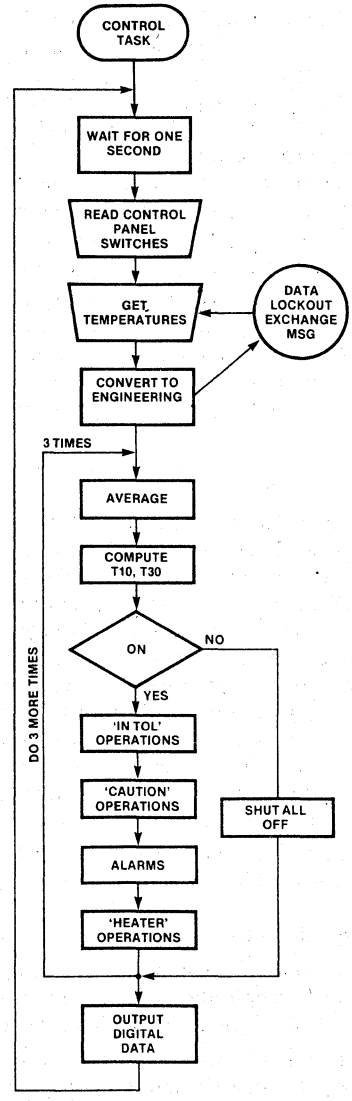


Figure 32. Control Task Flowchart

The number obtained from the analog converter provides us with a value which is proportional to the temperature of the oven. Our next step is to convert this number into engineering units. Unfortunately, the voltage and temperature are not related in a linear fashion since the thermistor is a nonlinear device. We will have to develop a technique to obtain a corrected value. For the purposes of this application note and in an attempt to keep the application as simple as possible, we have chosen to utilize a single table look-up to perform this conversion. Alternatives might have been to utilize FORTRAN routines to mathematically perform the conversion or to have separate tables for each oven. Once the conversion has been made, we must return a message to the data lockout exchange to allow other tasks access to the data.

Because we must deal with four ovens, the operations related to each individual oven must be performed four times, once for each. This is easily handled as we will see, since PL/M is a block structured language. Our flowchart need only remind us that the operations need be done four times.

The next step has been defined as performing some digital filtering of the temperature by averaging the current temperature with the temperature of one second ago. This filtered value will be used to perform subsequent computations and to make future decisions.

We have defined earlier in our definition of the control algorithm that we would use a derivative control. We have chosen to project the temperature ahead for a period of 10 and 30 seconds. We must calculate the rate of change and the temperatures in 10 and 30 seconds so that this data will be available when needed.

Now that the calculations have been made to determine numeric values required for the decision making process, we must begin the process of determining the status of each indicator and oven heater. A test will be made of the oven run switch and if it is found to be turned off, we will turn off all indicators and the oven heater associated with that oven. If the switch is found to be turned on, we will set the status of the "in tolerance", "caution", and "alarm" indicators according to our oven control algorithm. The oven heater will be turned on or off according to the projected temperature in 30 seconds.

Rather than output the individual oven indicator and heater data four times (once for each oven), we will perform the computations associated with making the decision four times (this saves code since we can use the same program steps with only pointers being exchanged). At the end of this time, a single operation will output the data to all ovens and indicators at the same time. Outputting to a computer port will actually cause the device to turn on or off according to whether the output bit is a one or zero.

We will then return to the beginning of our task to wait until another second elapses before we again perform the indicated functions.

Control Task Source Coding — The coding of our tasks is a straightforward procedure once we have prepared a flowchart. Since we are using PL/M-80 and RMX/80, the coding sequence for a task will be as follows:

1. Define any variables or structures which will be used in the module. This involves providing information defining variables as being either an 8 or 16-bit variable and declaring if that variable is to be a part of the task being coded or is to be found in some other task. If any arrays or structures are to be used, they must also be defined. Finally, if any program locations are to be used, they must be declared.
2. The task must be initialized. That is to say that any assumptions which will be made as to initial data values in subsequent instructions must be initially forced to this initial value.
3. The actual task must be coded to match the operations called out in the flowchart.

We will look at some examples of this coding process using the control task flowchart. The complete listing of this module and all modules actually used to provide the oven control system can be found in Appendix C.

At first glance, it would seem that the listing is extremely complex, but as we will see it is made up of straightforward pieces. The listing is made up of three parts as we have mentioned above when defining the steps required to generate a program. The first part (line numbers 1 through 50) is used to define parameters, variables, and external elements. The general types of elements making up this portion fall into typical categories. The first general category consists of DECLARE statements. Examples of typical lines will help explain their meanings (when actually developing the program, this first section was created piecemeal by

making an entry when it was found that a need for that term existed as the execution code in sections two and three were written).

Examples of the "declare" statement are shown below. For example, on line 11 we find:

```
11  1     Declare (n,k) byte;
```

This means that the variables "n" and "k" are being defined as terms which represent numbers or data which is one byte or 8 bits wide. The "11" is the program line number, and the "1" indicates that we are in the first level of nesting.

We can also see the use of the "literal" expressions such as used in line 4. The expression:

```
4  1     DECLARE FALSE LITERALLY '00H';
```

means that we are creating a new instruction called "false" and that its meaning is to be interpreted by the compiler as being equivalent to the value of zero.

Rather than dwell on the declaration, let us move on to the coding process which was used to generate the actual program. Keep in mind that the use of PL/M-80 requires that all terms used be declared in the program module. Refer to the *PL/M-80 Programming Manual* (order number 9800268B) for a full description of the PL/M language.

Program Initialization — The initialization portion of the program can be found on lines 51 through 59 of the control task program listing. This section is used to initialize data and to provide known entry conditions before we enter the repetitive program loop. This code is only executed when the system is reset or when the power is turned on. The control task requires two types of initializations; one to initialize the computer's output port and the other to set up the A/D converter. The requirements for each can be found in the RMX/80 User's Guide and the *iSBC 80/30 Single Board Computer Hardware Reference Manual* (order number 9800611A). Actual instruction examples are given in these manuals for the initialization operations.

Program Body — The program which actually provides the control operations can be found on lines 60 through 126 of the program listing for the control task. It has been divided into sections which correspond directly to the flowchart that was prepared earlier. Most instructions in PL/M-80 language follow closely the English structure which describes what is being done. The exceptions generally follow definite predefined formats. The format such as used on line 61 to wait for one second to elapse is an example of one such exception. Any time we desire to wait for a definite time period, we use an instruction of the form:

```
MSG$PTR = RQWAIT (.DUMMY$EXCH, TIME DELAY);
```

Whatever time delay we wish to use is expressed in increments of 50 msec time periods. Our example requires a time delay of one second so we will use the delay notation of 1.0/0.050 = 20 time units (this command is actually calling upon the RMX/80 executive to handle the delay).

The oven enable switch data has been defined by us to be routed by the hardware to the computer port "EA" which converts to a decimal number, 234. If we define an internal memory location for this data and call it BLOCK0, then we can get the oven switch data by using an input statement. Since the data sense is inverted through the hardware, we can provide meaningful internal data if the signal is re-inverted as it is loaded into memory. The instruction on line 62 of the control task listing performs this task.

We are now ready to get the analog data from the A/D converter. Our flowchart shows that we must lock out the other tasks from access to the temperature data during this time period, so we must first remove the enable message from the exchange in which it is stored. Messages are removed from an exchange by using an instruction of the form:

```
STORAGE = RQWAIT (EXCHANGE NAME,0)
```

Line 63 of the program listing means that we will get a message from our storage exchange which is called "Temp$lockout$exch" and store it in a memory storage area called "Lockout". Now, no other task can get a message from this exchange since it is empty, so it is permissible to operate on the temperature data. (Note how similar this command is to the one used to wait for a delay. Indeed, this is the same request for RMX/80, but it requests a time delay of zero.)

During the initialization, we built a message defining the characteristics of the analog signals and of the analog conversion board which we are using. Remember that we have indicated that the task of getting this data from the board is provided to us by one of RMX/80's predefined drivers. All that is necessary at this time is to inform that driver of our desire to get data, then wait until it has done its job and the data is available for us. The actual communication between our applications task and the analog driver is done using the idea of an exchange similar to that we have used to lockout the data.

A flowchart can now be prepared showing the steps required to implement the CRT update task. This flowchart is shown in Figure 34. The coding of the program to support this task can be found in Appendix C. The development is identical with that which we described in the sections regarding the control task. Again, the software is divided into three parts, the declaration statements from line 1 to 81, the initialization on lines 82 to 87, and the actual task code on line 88 to 207.

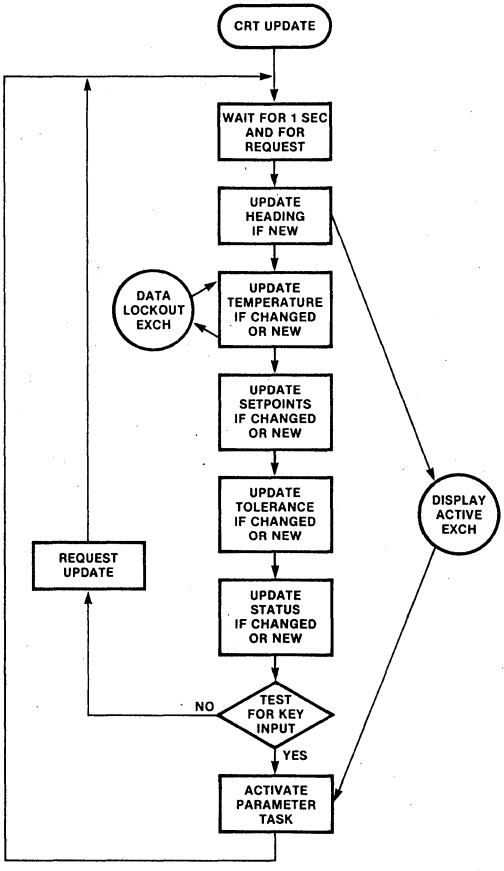


Figure 34. CRT Status Flowchart

A technique to exit from the CRT update mode and to get into a mode which will allow modification of the parameters has been introduced into the program and the display format. This is in the form of a message on the botton of the screen requesting the entry of an escape character to adjust setpoints. The software has been written in such a manner as to test for a character inut from the keyboard and if one is found corresponding to that character, the update task will allow the parameter update task to take control of the terminal (lines 190 to 204 of the listing).

Parameter Update Task

The parameter update task is used to actually allow the modification of the setpoints and the tolerances associated with each oven. A second use of the task is to provide a tool for establishing the zero offset associated with each analog channel so that an offset into the temperature linearization table can be computed by the control task.

Figure 35 shows the flowchart which describes the steps required to perform these operations. When the task has been completed, we will return to the CRT update task.

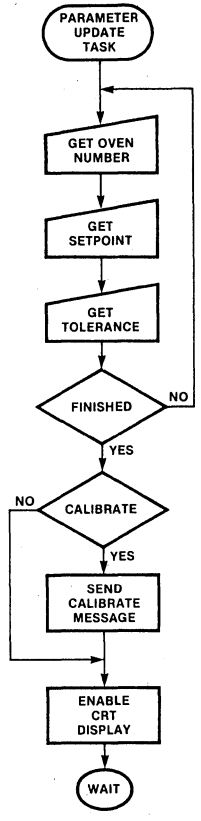


Figure 35. Parameter Update Flowchart

The program code for this task can be found in Appendix C and again follows the formats which we have discussed earlier. No attempt will be made in this document to provide a narrative of the listing since it follows the flowchart in development.

Support Programs

Three subprograms (procedures) have been written which provide functions which are common to the three tasks. This has been done to minimize repeating code segments thus saving as much memory as possible. These three subprograms support:

1. Conversion of a decimal string from the terminal into a binary number. This program is called ASC2BINARY and can be found in Appendix C.
2. Storage for common variables used by more than one task. These variables could easily have been included in other tasks but a purely arbitrary decision was made to include them in a separate module.
3. Conversion of binary numbers into a decimal string suitable for output to the terminal. This program is called DEC$REP and is found in Appendix C.

We now have completed the coding of the software to support our oven application. We must finish by combining all the software together to form a single loadable module.

VI. FINAL IMPLEMENTATION

When all code was linked and loaded to form an executable program module, it was found that t'.e system required 9,041 bytes of EPROM and 1,735 bytes of RAM. These values fall within our hardware capabilities and will rquire that we program and insert nine EPROMs into the EPROM expansion card.

The system can now be tested and installed to control the ovens of our application. The actual system described in this application note has been constructed and tested. It has been found to control the oven temperatures of four ovens and performs as we anticipated when we developed our control strategy earlier in this application note.

VII. CONCLUSION

We have shown how Intel's single board computers, industrial chassis, termination panels, and software can be configured to provide a solution to a typical control application. We have seen how the development of a solution to a control problem can proceed along a predetermined and logical path. Truly, the utilization of the microprocessors can lead to optimum and cost effective solutions to control applications.

We will send a message to the analog driver telling it what we want it to do, then we will wait until it sends a message back to one of our exchanges telling us that it is done. The format for sending a message to an exchange always follows the form:

CALL RQSEND (EXCHANGE NAME, MESSAGE NAME);

Line 64 of the listing shows that we have requested the input of the analog data since we have sent our message, Convert, to the analog driver's exchange which is called RQAIEX. We will wait until the operation is complete by using the line of code shown on the listing line 65. This is the same operation type that we used to get our message back providing a lockout earlier. The program will wait until a message is available before continuing.

The data must now be converted into engineering units. We earlier indicated that we would use a table lookup to perform the linearization, so we have included this table as a part of our program at line 50. The offset into the table corresponding to our temperature must be determined so that the correct value can be stored. Because we have four ovens, we will perform the operation four times with the data each time corresponding to the appropriate oven. These operations can be followed on lines 77 through 81 of the listing.

Lines 67 through 76 are used to establish an offset to be applied to the analog temperature data when the system is running. This program is only designed to be used during the start-up operations and is activated when a message containing a calibration request and current temperature is sent to its exchange.

The temperature lockout must be removed to enable other tasks to use this data. This is done on line 82 by sending the message back to the exchange used for intertask lockout communications.

The remainder of the program follows the flowchart and the operations can be followed using a flowchart and the listing. Each element of the flowchart corresponds to a block of code on the listing.

CRT Update Task Development

Earlier, we stated that the CRT update task would be used to allow the operator to view a "snapshot" of the four ovens. Let us turn our attention to developing the software which is required to accomplish this. We can begin by defining the elements which we feel should be displayed, then defining the format to actually be used with the CRT terminal.

Obviously, we need to provide the current temperature of each oven on our display screen. If we display the actual temperature, it seems reasonable to assume that we should also show the setpoint so that a determination can be made as to how well the system is performing. The control algorithm has been defined to use an allowable range to determine system outputs, so it would seen wise to also show this parameter. Finally, we should inform the viewer of the status of the oven so that he will realize that the reason an oven temperature is low is because the oven is off rather than an oven malfunction. Other items could be added if desired by the system designer, depending upon the total system requirements or the characteristics of the users.

We can now prepare a drawing of the CRT display to generate a layout of our desired characters and to generate an aesthetic display for viewing during operation. This drawing can be found in Figure 33.

Several techniques are available to output the required displays to the terminal. A decision must be made as to the frequency of screen updates; will we constantly refresh the data or do it only at certain intervals of time? If the terminal has the ability to disable the cursor, it makes sense to update data continuously. If the cursor cannot be disabled, its movement tends to be distracting, so the updates should be kept to a minimum. The terminal used for the application note did not have a disable feature, so we will make the decision to only update the screen once each second.

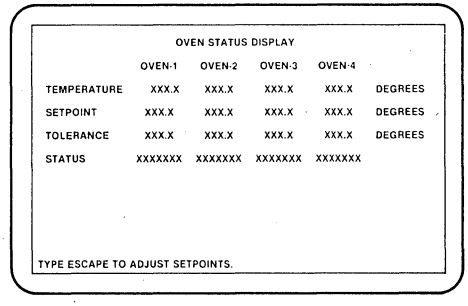


Figure 33. CRT Status Display Layout

The decision to delay updates leads us to make another decision regarding the screen updates. If we only update a line which has data which has changed since the last update, the cursor movements will be kept at a minimum since it is unlikely that all parameters will ever change each second.

APPENDIX A
SELECTED DATA SHEETS

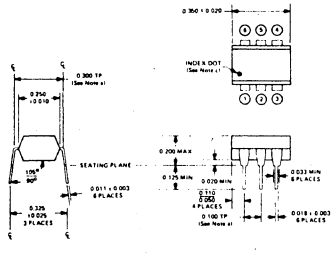
TYPES TIL113, TIL119 OPTO-COUPLERS

BULLETIN NO. DL-S 7312032, NOVEMBER 1973

- Gallium Arsenide Diode Infrared Source Optically Coupled to a Silicon N-P-N Darlington-Connected Phototransistor
- High Direct-Current Transfer Ratic . . . 300% Minimum at 10 mA
- Base Lead Provided for Conventional Transistor Biasing
- High-Voltage Electrical Isolation . . . 1500-Volt Rating
- Plastic Dual-In-Line Package
- Typical Applications Include Remote Terminal Isolation, SCR and Triac Triggers, Mechanical Relays, and Pulse Transformers

mechanical data

The package consists of a gallium arsenide infrared-emitting diode and an n-p-n silicon darlington-connected phototransistor mounted on a 6-lead frame encapsulated within an electrically nonconductive plastic compound. The case will withstand soldering temperature with no deformation and device performance characteristics remain stable when operated in high humidity conditions. Unit weight is approximately 0.52 grams.



NOTES:
a. Leads are within 0.005 radius of true position (TP) at the gauge plane with maximum material condition and unit installed.
b. All dimensions are in inches unless otherwise noted.
c. Pin 1 identified by index dot.
d. Terminal connections:
   1. Anode } Infrared-emitting diode
   2. Cathode }
   3. No internal connection
   4. Emitter
   5. Collector } Phototransistor
   6. Base (For TIL119, make no external connection) }

absolute maximum ratings at 25°C free-air temperature (unless otherwise noted)

Input-to-Output Voltage	±1.5 kV
Collector-Base Voltage (TIL113)	30 V
Collector-Emitter Voltage (See Note 1)	30 V
Emitter-Collector Voltage	7 V
Emitter-Base Voltage (TIL113)	7 V
Input-Diode Reverse Voltage	3 V
Input-Diode Continuous Forward Current at (or below) 25°C Free-Air Temperature (See Note 2)	100 mA
Continuous Power Dissipation at (or below) 25°C Free-Air Temperature:	
Infrared-Emitting Diode (See Note 3)	150 mW
Phototransistor (See Note 4)	150 mW
Total (Infrared-Emitting Diode plus Phototransistor, See Note 5)	250 mW
Storage Temperature Range	−55°C to 150°C
Lead Temperature 1/16 Inch from Case for 10 Seconds	260°C

NOTES: 1. This value applies when the base-emitter diode is open-circuited.
2. Derate linearly to 100°C free-air temperature at the rate of 1.33 mA/°C.
3. Derate linearly to 100°C free-air temperature at the rate of 2 mW/°C.
4. Derate linearly to 100°C free-air temperature at the rate of 2 mW/°C.
5. Derate linearly to 100°C free-air temperature at the rate of 3.33 mW/°C.

electrical characteristics at 25°C free-air temperature

PARAMETER		TEST CONDITIONS†			TIL113			TIL119			UNIT
					MIN	TYP	MAX	MIN	TYP	MAX	
$V_{(BR)CBO}$	Collector-Base Breakdown Voltage	I_C = 10 μA,	I_E = 0,	I_F = 0	30						V
$V_{(BR)CEO}$	Collector-Emitter Breakdown Voltage	I_C = 1 mA,	I_B = 0,	I_F = 0	30			30			V
$V_{(BR)EBO}$	Emitter-Base Breakdown Voltage	I_E = 10 μA,	I_C = 0,	I_F = 0	7						V
$V_{(BR)ECO}$	Emitter-Collector Breakdown Voltage	I_E = 10 μA,	I_F = 0					7			V
$I_{C(on)}$	On-State Collector Current	V_{CE} = 1 V,	I_B = 0,	I_F = 10 mA	30	100					mA
		V_{CE} = 2 V,	I_F = 10 mA					30	160		
$I_{C(off)}$	Off-State Collector Current	V_{CE} = 10 V,	I_B = 0,	I_F = 0			100			100	nA
h_{FE}	Transistor Static Forward Current Transfer Ratio	V_{CE} = 1 V,	I_C = 10 mA,	I_F = 0		15,000					
V_F	Input Diode Static Forward Voltage	I_F = 10 mA					1.5			1.5	V
$V_{CE(sat)}$	Collector-Emitter Saturation Voltage	I_C = 125 mA,	I_B = 0,	I_F = 50 mA			1				V
		I_C = 10 mA,	I_F = 10 mA							1	
r_{IO}	Input-to-Output Internal Resistance	V_{in-out} = ±1.5 kV, See Note 6			10^{11}			10^{11}			Ω
C_{io}	Input-to-Output Capacitance	V_{in-out} = 0,	f = 1 MHz,	See Note 6		1	1.3		1	1.3	pF

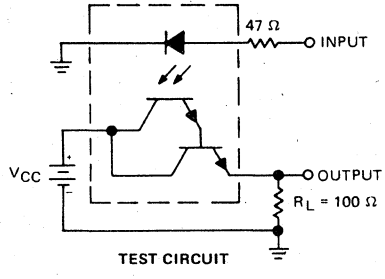
NOTE 6: These parameters are measured between both input-diode leads shorted together and all the phototransistor leads shorted together.

† References to the base are not applicable to the TIL119.

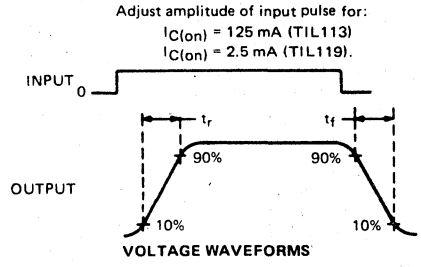
switching characteristics at 25°C free-air temperature

PARAMETER		TEST CONDITIONS		TIL113			TIL119			UNIT
				MIN	TYP	MAX	MIN	TYP	MAX	
t_r	Rise Time	V_{CC} = 15 V,	$I_{C(on)}$ = 125 mA,		50					μs
t_f	Fall Time	R_L = 100 Ω,	See Figure 1		50					
t_r	Rise Time	V_{CC} = 10 V,	$I_{C(on)}$ = 2.5 mA,					50		μs
t_f	Fall Time	R_L = 100 Ω,	See Figure 1					50		

PARAMETER MEASUREMENT INFORMATION



TEST CIRCUIT



VOLTAGE WAVEFORMS

NOTES: a. The input waveform is supplied by a generator with the following characteristics: Z_{out} = 50 Ω, $t_r \leq$ 15 ns, duty cycle ≈ 1%, t_w = 100 μs.

b. The output waveform is monitored on an oscilloscope with the following characteristics: $t_r \leq$ 12 ns, $R_{in} \geq$ 1 MΩ, $C_{in} \leq$ 20 pF.

FIGURE 1–SWITCHING TIMES

TYPICAL CHARACTERISTICS

TIL113
COLLECTOR CURRENT
vs
COLLECTOR-EMITTER VOLTAGE

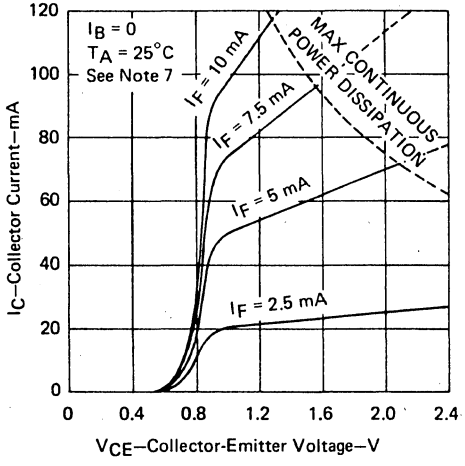


FIGURE 2

TIL119
COLLECTOR CURRENT
vs
COLLECTOR-EMITTER VOLTAGE

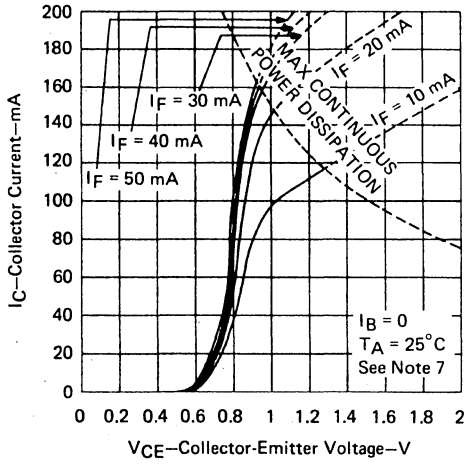


FIGURE 3

TIL113
COLLECTOR CURRENT
vs
INPUT-DIODE FORWARD CURRENT

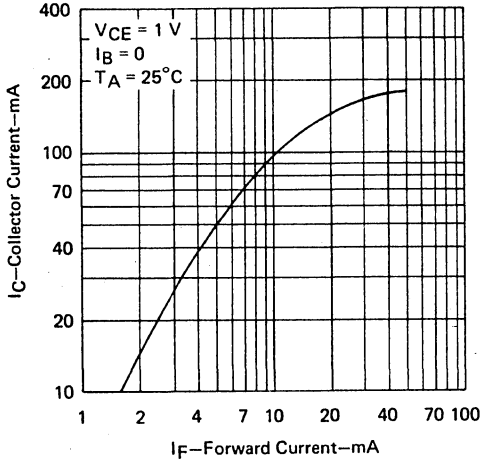


FIGURE 4

OFF-STATE COLLECTOR CURRENT
vs
FREE-AIR TEMPERATURE

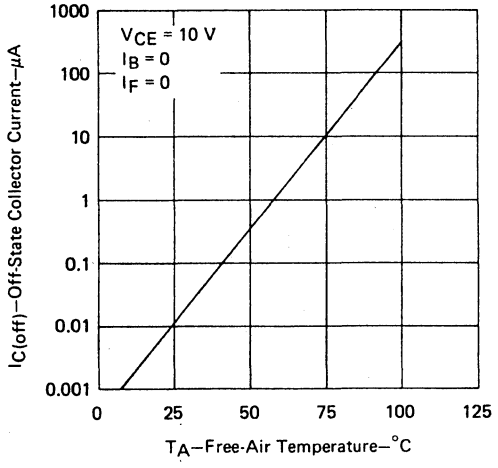


FIGURE 5

NOTE 7: Pulse operation of input diode is required for operation beyond limits shown by dotted line.

TYPICAL CHARACTERISTICS

TIL113
RELATIVE COLLECTOR-EMITTER SATURATION VOLTAGE
vs
FREE-AIR TEMPERATURE

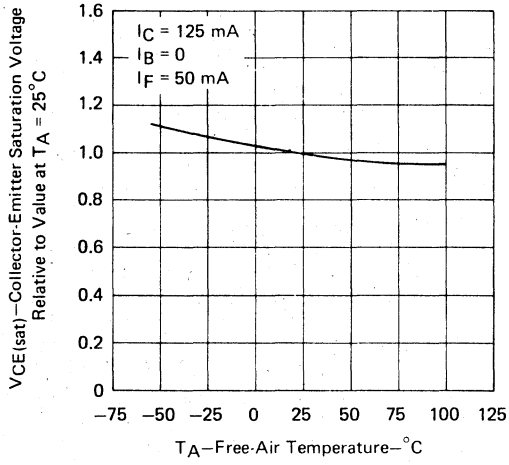


FIGURE 6

TIL113
TRANSISTOR STATIC FORWARD CURRENT TRANSFER RATIO
vs
COLLECTOR CURRENT

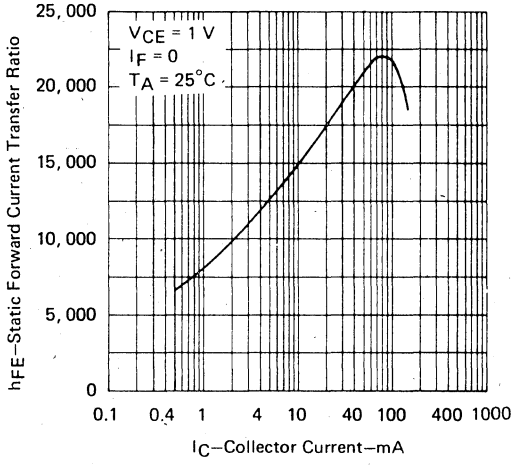


FIGURE 7

INPUT DIODE FORWARD CONDUCTION CHARACTERISTICS

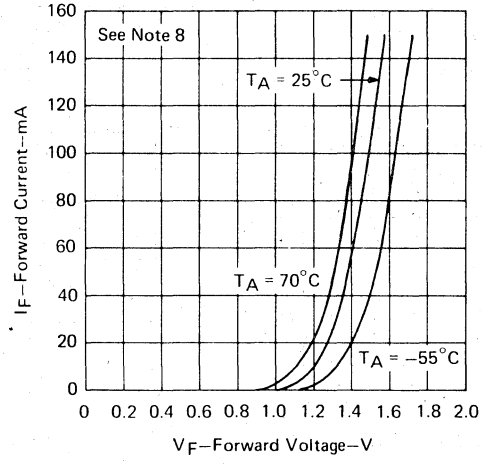


FIGURE 8

NOTE 8: This parameter was measured using pulse techniques. t_w = 1 ms, duty cycle ≤ 2%.

OPTO 22

I/O Module Detail
Electrical Specifications

AC INPUT MODULES	MODEL IAC5	MODEL IAC15	MODEL IAC24	MODEL IAC5-A	MODEL IAC15-A	MODEL IAC24-A
AC INPUT LINE VOLTAGE	95 to 130 VAC			180 to 280 VAC		
INPUT CURRENT AT RATED LINE	10 ma					
ISOLATION INPUT TO OUTPUT	2500 Volt RMS					
INPUT ALLOWED FOR NO OUTPUT	1.5 ma					
TURN ON TIME	20 Millisecond Maximum					
TURN OFF TIME	20 Millisecond Maximum					
OUTPUT TRANST. BREAKDOWN	30 Volts DC					
OUTPUT CURRENT	25 ma					
OUTPUT LEAKAGE 30VDC, NO. INPUT	100 Microamp Maximum					
OUTPUT VOLTAGE DROP	.4 Volts at 25 ma Load					
LOGIC SUPPLY VOLTAGE DC	4.5 to 6 V	12 to 18 V	20 to 30 V	4.5 to 6 V	12 to 18 V	20 to 30 V
LOGIC SUPPLY CURRENT	12 ma	15 ma	18 ma	12 ma	15 ma	18 ma

AC OUTPUT MODULES	MODEL OAC5	MODEL OAC15	MODEL OAC24	MODEL OAC5-A	MODEL OAC15-A	MODEL OAC24-A
LINE VOLTAGE	12 to 140 VAC			24 to 280 VAC		
CURRENT RATING	3 Amps①					
1-CYCLE SURGE	55 Amps Peak					
SIGNAL INPUT RESISTANCE	220 Ohm	1K Ohm	2.2K Ohm	220 Ohm	1K Ohm	2.2K Ohm
SIGNAL PICKUP VOLTS DC	3V 8V Ald.*	9V 16V Ald.*	18V 32V Ald.*	3V 8V Ald.*	9V 16V Ald.*	18V 32V Ald.*
SIGNAL DROPOUT VOLTS DC	1 Volt					
PEAK REPETITIVE VOLTAGE	400V			500 Volts		
MAXIMUM CONTACT DROP	1.6V					
OFF STATE LEAKAGE	5 ma RMS					
MINIMUM LOAD CURRENT	20 ma					
ISOLATION INPUT TO OUTPUT	2500 Volts RMS					
CAPACITANCE INPUT TO OUTPUT	8 Pf					
STATIC DV/DT	200 Volts/Microsecond Min					
COMMUTATING DV/DT	Built in snubber (will commutate .5 power factor loads)					

*Allowed

DC INPUT MODULES	MODEL IDC5	MODEL IDC15	MODEL IDC24
INPUT LINE VOLTAGE	10-32 VDC		
INPUT CURRENT	32 ma at 32V		
ISOLATION INPUT TO OUTPUT	2500 Volt RMS		
CAPACITANCE INPUT TO OUTPUT	8 Pf		
INPUT ALLOWED FOR NO OUTPUT	2 ma		
TURN ON TIME	5 Millisecond Max		
TURN OFF TIME	5 Millisecond Max		
OUTPUT TRANST. BREAKDOWN	30 Volts DC		
OUTPUT CURRENT	25 ma		
OUTPUT LEAKAGE 30 VDC NO INPUT	100 Microamps Max		
OUTPUT VOLTAGE DROP	.4 Volt at 25 ma		
LOGIC SUPPLY VOLTAGE	4.5 to 6V	12 to 18V	20 to 30V
LOGIC SUPPLY CURRENT	12 ma	15 ma	18 ma

DC OUTPUT MODULES	MODEL ODC5	MODEL ODC15	MODEL ODC24
LOAD VOLTAGE RATING	60V DC		
OUTPUT CURRENT RATING	3 Amps①		
OFF STATE LEAKAGE	1 ma Max		
ISOLATION INPUT TO OUTPUT	2500 V RMS		
SIGNAL PICK UP VOLTAGE	3V 8V Ald.*	9V 18V Ald.*	18V 28V Ald.*
SIGNAL DROP OUT VOLTAGE	1Volt		
SIGNAL INPUT RESISTANCE	220 Ohm	1K Ohm	2.2K Ohm
1 SECOND SURGE	5 Amps		
TURN ON TIME	500 Microsecond		
TURN OFF TIME	2.5 Millisecond		

*Allowed
①Derate .033 Amps per degree C from 20° C

opto 22

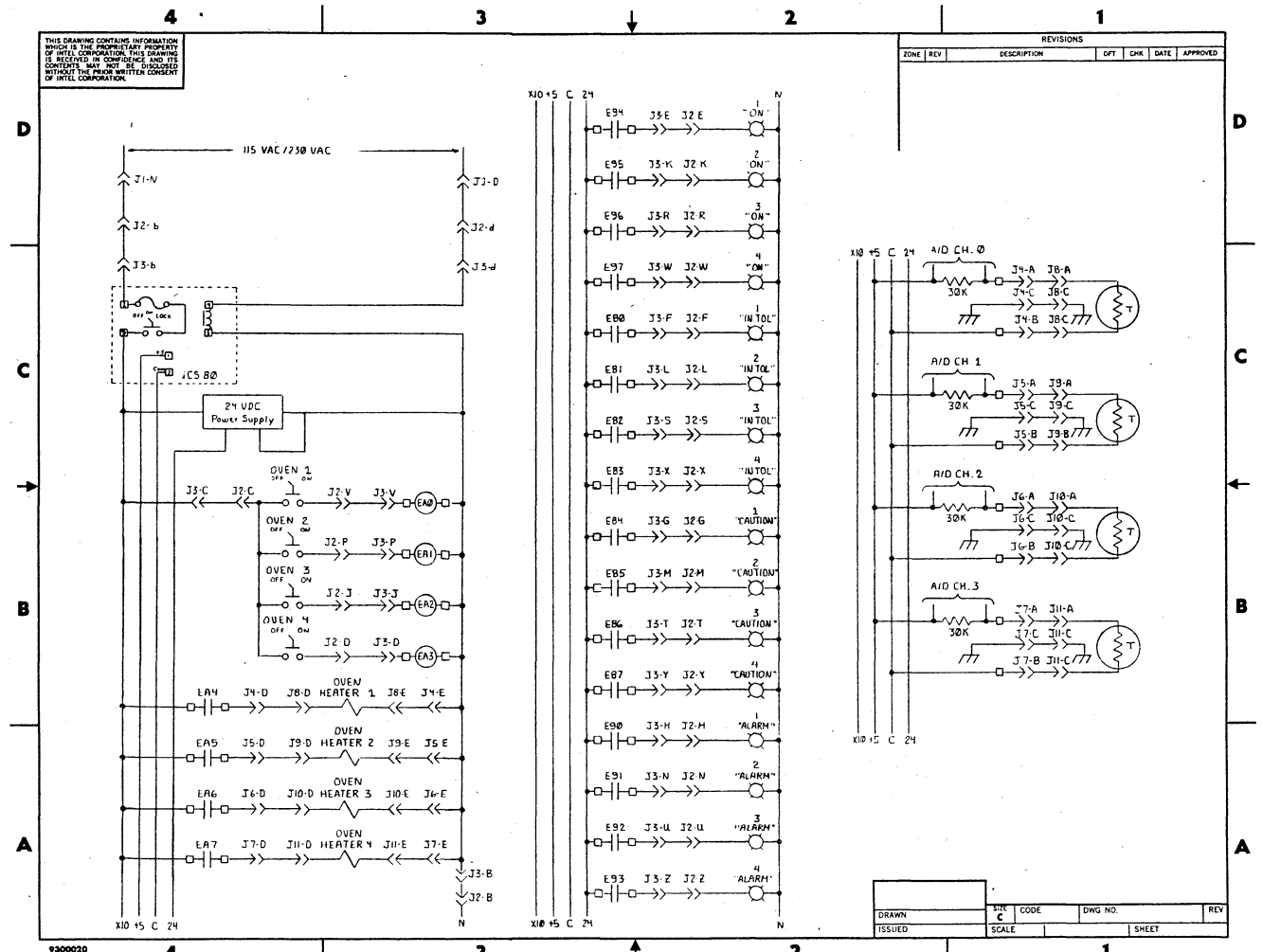
High Voltage DC Output Modules

DC OUTPUT MODULES	MODEL ODC5-A	MODEL ODC15-A	MODEL ODC24-A
LOAD VOLTAGE RATING	200V DC		
OUTPUT CURRENT RATING	1 Amps		
OFF STATE LEAKAGE	2 ma Max		
ISOLATION INPUT TO OUTPUT	2500 V RMS		
SIGNAL PICK UP VOLTAGE	3V 8V Ald.*	9V 18V Ald.*	18V 28V Ald.*
SIGNAL DROP OUT VOLTAGE	1Volt		
SIGNAL INPUT RESISTANCE	220 Ohm	1K Ohm	2.2K Ohm
1 SECOND SURGE	5 Amps		
TURN ON TIME	500 Microsecond		
TURN OFF TIME	2.5 Millisecond		

*Allowed

Fast Switching DC Input Modules

DC INPUT MODULES	MODEL IDC5-B	MODEL IDC15-B	MODEL IDC24-B
INPUT LINE VOLTAGE	4-16 VDC		
INPUT CURRENT	14 ma at 5V		
ISOLATION INPUT TO OUTPUT	2500 Volt RMS		
CAPACITANCE INPUT TO OUTPUT	8 Pf		
INPUT ALLOWED FOR NO OUTPUT	1 Volt		
TURN ON TIME	50 Microsecond Max		
TURN OFF TIME	100 Microsecond Max		
OUT TRANSISTOR BREAKDOWN	30 Volts DC		
OUTPUT CURRENT	25 ma		
OUTPUT LEAKAGE 30 VDC NO INPUT	100 Microamps Max		
OUTPUT VOLTAGE DROP	.4 Volt at 25 ma		
LOGIC SUPPLY VOLTAGE	4.5 to 6V	12 to 18V	20 to 30V
LOGIC SUPPLY CURRENT	12 ma		


THIS DRAWING CONTAINS INFORMATION WHICH IS THE PROPRIETARY PROPERTY OF INTEL CORPORATION. THIS DRAWING IS RECEIVED IN CONFIDENCE AND ITS CONTENTS MAY NOT BE DISCLOSED WITHOUT THE PRIOR WRITTEN CONSENT OF INTEL CORPORATION.
REVISIONS
ZONE | REV | DESCRIPTION | DFT | CHK | DATE | APPROVED
115 VAC / 230 VAC
J1-N
J2-b
J3-b
J1-D
J2-d
J3-d
OFF LOCK
1CS 8Ø
24 VDC Power Supply
OVEN 1
OVEN 2
OVEN 3
OVEN 4
J3-C J2-C J2-V J3-V EAØ
J2-P J3-P EA1
J2-J J3-J EA2
J2-D J3-D EA3
EA4 J4-D J8-D OVEN HEATER 1 J8-E J4-E
EA5 J5-D J9-D OVEN HEATER 2 J9-E J5-E
EA6 J6-D J10-D OVEN HEATER 3 J10-E J6-E
EA7 J7-D J11-D OVEN HEATER 4 J11-E J7-E
J3-B
J2-B
X10 +5 C 24
E94 J3-E J2-E "ON" 1
E95 J3-K J2-K "ON" 2
E96 J3-R J2-R "ON" 3
E97 J3-W J2-W "ON" 4
E80 J3-F J2-F "IN TOL" 1
E81 J3-L J2-L "IN TOL" 2
E82 J3-S J2-S "IN TOL" 3
E83 J3-X J2-X "IN TOL" 4
E84 J3-G J2-G "CAUTION" 1
E85 J3-M J2-M "CAUTION" 2
E86 J3-T J2-T "CAUTION" 3
E87 J3-Y J2-Y "CAUTION" 4
E90 J3-H J2-H "ALARM" 1
E91 J3-N J2-N "ALARM" 2
E92 J3-U J2-U "ALARM" 3
E93 J3-Z J2-Z "ALARM" 4
A/D CH. Ø
A/D CH. 1
A/D CH. 2
A/D CH. 3
30K
J4-A J8-A
J4-C J8-C
J4-B J8-C
J5-A J9-A
J5-C J9-C
J5-B J9-B
J6-A J10-A
J6-C J10-C
J6-B J10-C
J7-A J11-A
J7-C J11-C
J7-B J11-C
T
DRAWN
ISSUED
SIZE C
CODE
DWG NO.
REV
SCALE
SHEET
9300020

```
                $TITLE ('CONTROL TASK')
               /*******************************************************
               * This task handles the control and monitoring of      *
               * four oven chambers.                                  *
               *******************************************************/
1              CONTROLSTASK$MODULE:
               Do;
2     1        DECLARE EXCHANGE$DESCRIPTOR LITERALLY 'STRUCTURE (
                MESSAGE$HEAD ADDRESS,
                MESSAGE$TAIL ADDRESS,
                TASK$HEAD ADDRESS,
                TASK$TAIL ADDRESS,
                EXCHANGE$LINK ADDRESS)';
3     1        DECLARE TRUE LITERALLY '0FFH';
4     1        DECLARE FALSE LITERALLY '00H';
5     1        DECLARE BOOLEAN LITERALLY 'BYTE';
6     1        DECLARE FOREVER LITERALLY 'WHILE 1';
7     1        DECLARE MSG$HDR LITERALLY '
                LINK ADDRESS,
                LENGTH ADDRESS,
                TYPE BYTE,
                HOME$EX ADDRESS,
                RESP$EX ADDRESS';
8     1        DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE(
                MSG$HDR,
                REMAINDER(1) BYTE)';
               /* AIMSG.ELT - ANALOG INPUT REQUEST MESSAGE FORMAT */
9     1        DECLARE AIMSG LITERALLY 'STRUCTURE(
                              MSG$HDR,
                              STATUS ADDRESS,
                              BASE$PTR ADDRESS,
                              CHANNEL$GAIN ADDRESS,
                              ARRAY$PTR ADDRESS,
                              COUNT ADDRESS,
                              ACTUAL$COUNT ADDRESS)';
               /* AITYP.ELT - ANALOG INPUT MESSAGE TYPES */
10    1        DECLARE AIREP LITERALLY '30',
                       AISQS LITERALLY '31',
                       AISQV LITERALLY '32',
                       AIRAN LITERALLY '33';
11    1          Declare  (n,k) byte;
12    1        Declare (MSG$PTR,LOCKOUT) address;
13    1          Declare (BLOCK0,BLOCK1,BLOCK2,BLOCK3) byte external;
14    1        Declare TOLERANCE(4) address external;
15    1        Declare TEMP(4) address external;
16    1        Declare SETPOINT(4) address external;
17    1        Declare T$AVERAGE(4) address;
18    1        Declare T$LAST(4) address;
19    1        Declare T$LAST$AVERAGE(4) address;
20    1        Declare T$t5(4) address;
21    1        Declare T$t10(4) address;
22    1        Declare STATUS(4) byte external;
23    1          Declare CRT$DISPLAY$LOCK(5) address external;
```

```
24   1       Declare TEMP$CALIBRATE(5) address external;
25   1       Declare DUMMY$EXCH(5) address external;
26   1       Declare TEMP$LOCKOUT$EXCH(5) address external;
27   1       Declare RQAIEX(5) address external;
28   1       Declare ASD$EXCH(5) address external;
29   1       Declare CONSTANT$LOCKOUT$EXCH(5) address external;
30   1       Declare CRT$STATUS$EXCH(5) address external;
31   1         Declare ALARM$MSG structure (MSG$HDR);
32   1       Declare CONVERT ai$msg;
         /* This term is used to convey initial temperatures */
33   1       Declare CAL$TEMP based MSG$PTR structure (
                MSG$HDR,
                CAL address );
34   1         RQWAIT:
                Procedure (EXCH,MESSAGE) address external;
35   2           Declare (EXCH,MESSAGE) address;
36   2       end RQWAIT;
37   1         RQSEND:
                Procedure (EXCH,MESSAGE) external;
38   2           Declare (EXCH,MESSAGE) address;
39   2       end RQSEND;
40   1         RQACPT:
                Procedure (EXCH) address external;
41   2           Declare EXCH address;
42   2       end RQACPT;
43   1         Declare OVEN$IN$TOL(4) byte data (
                01H,02H,04H,08H );
44   1       Declare OVEN$CAUTION(4) byte data (
                10H,20H,40H,80H );
45   1       Declare OVEN$DANGER(4) byte data (
                01H,02H,04H,08H );
46   1       Declare OVEN$ON$MASK(4) byte data (
                01H,02H,04H,08H );
47   1       Declare OVEN$HEATER(4) byte data (
                10H,20H,40H,80H );
48   1       Declare OVEN$RUN(4) byte data (
                10H,20H,40H,80H );
49   1       Declare OFFSET(4) address;
50   1       Declare TABLE(256) address data (
                200,201,202,203,204,205,206,207,208,209,
                209,210,211,212,213,214,215,216,217,218,
                219,220,221,222,223,224,225,226,227,228,
                229,230,231,232,233,235,236,237,238,239,
                240,241,243,244,245,247,248,249,250,251,
                252,254,256,257,258,259,260,261,263,265,
                266,267,268,269,270,271,273,274,276,278,
                279,280,282,284,285,287,288,289,290,291,
                293,295,296,298,299,300,302,304,305,307,
                308,309,310,312,314,316,318,320,322,324,
                326,328,330,332,334,336,338,340,342,344,
                346,348,350,352,354,356,358,360,362,364,
                366,368,370,372,374,376,378,380,382,385,
                388,390,392,395,398,400,402,405,407,410,
                412,415,418,420,423,426,428,430,433,436,
                439,441,444,447,451,454,457,460,463,466,
```

```
                470,473,476,480,484,488,492,496,500,504,
                507,511,515,519,523,527,531,535,540,545,
                550,555,560,565,570,575,580,585,590,595,
                600,605,610,615,620,625,630,635,640,645,
                650,655,660,665,670,675,680,685,690,695,
                700,705,710,715,720,725,730,735,740,745,
                750,000,000,000,000,000,000,000,000,000,
                000,000,000,000,000,000,000,000,000,000,
                000,000,000,000,000,000,000,000,000,000,
                000,000,000,000,000,000                    );


                /* Initialization of control task */
51    1         CONTROL$TASK:
                Procedure public;
52    2           Output(235)=81H;
53    2           CONVERT.BASE$PTR=0F700H;
54    2           CONVERT.LENGTH=21;
55    2           CONVERT.TYPE=AISQS;
56    2           CONVERT.RESP$EX=.A$D$EXCH;
57    2           CONVERT.CHANNEL$GAIN=0;
58    2           CONVERT.ARRAY$PTR=.TEMP;
59    2           CONVERT.COUNT=4;
60    2           Do forever;
                  /* Wait for one second to elapse */
61    3             MSG$PTR=RQWAIT (.DUMMY$EXCH,20);
                  /* Bring in data from switches */
62    3             BLOCK0=NOT INPUT(234);
                  /* Lockout temperature storage areas for update */
63    3             LOCKOUT=RQWAIT (.TEMP$LOCKOUT$EXCH,0);
                  /* Get raw data from analog converter */
64    3             Call RQSEND (.RQAIEX,.CONVERT);
65    3             MSG$PTR=RQWAIT(.A$D$EXCH,0);
                  /* Temperature calibrate procedure */
66    3             MSG$PTR=RQACPT(.TEMP$CALIBRATE);
67    3             If MSG$PTR <> 0
                    then do;
69    4                 k=0;
70    4                 Do while (TABLE(k)<>CALTEMP.CAL AND
                                k<255);
71    5                   k=k+1;
72    5                 end;
73    4                 Do n=0 to 3;
74    5                   OFFSET(n)=(TEMP(n)/16)-k;
75    5                 end;
76    4             end;
                  /* Convert data into engineering units */
77    3             Do n=0 to 3;
78    4                 If ((TEMP(n)/16)-OFFSET(n))>255
                        then TEMP(n)=0;
80    4                 else TEMP(n)=TABLE((TEMP(n)/16)-OFFSET(n));
81    4             end;
                  /* Release lockout of temperatures */
82    3             Call RQSEND (.TEMP$LOCKOUT$EXCH,LOCKOUT);
                  /* Compute average temperature */
```

```
82   3          Do n=0 to 3;
84   4             T$AVERAGE(n)=(T$LAST(n)+TEMP(n))/2;
                /* Project temperatures into the future */
85   4             If T$AVERAGE(n)>=T$LAST$AVERAGE(n)
                    then do;
87   5               T$t5(n)=((T$AVERAGE(n)-T$LAST$AVERAGE(n))*5)
                          +T$LAST$AVERAGE(n);
88   5               T$t10(n)=((T$AVERAGE(n)-T$LAST$AVERAGE(n))*10)
                          +T$LAST$AVERAGE(n);
89   5             end;
90   4               else do;
91   5               T$t5(n)=T$LAST$AVERAGE(n)-((T$LAST$AVERAGE(n)
                          -T$AVERAGE(n))*5);
92   5               T$t10(n)=T$LAST$AVERAGE(n)-((T$LAST$AVERAGE(n)
                          -T$AVERAGE(n))*10);
93   5             end;
                /* Update stored data */
94   4             T$LAST$AVERAGE(n)=T$AVERAGE(n);
95   4             T$LAST(n)=TEMP(n);
                /* Test for active oven */
96   4             MSC$PTR=RQWAIT (.CONSTANT$LOCKOUT$EXCH,0);
97   4             If (((BLOCK0 AND OVEN$ON$MASK(n))<>0)
                    AND (TEMP(n)<>0))
                    then do;
99   5                 STATUS(n)=7;
100  5                 BLOCK2=BLOCK2 OR OVEN$RUN(n);
                /* Test for an intolerance condition */
101  5                 If SETPOINT(n)-TOLERANCE(n) < TEMP(n) AND
                         SETPOINT(n)+TOLERANCE(n) > TEMP(n)
                      then do;
103  6                     STATUS(n)=7;
104  6                     BLOCK1=BLOCK1 OR OVEN$IN$TOL(n);
105  6                   end;
106  5                   else BLOCK1=BLOCK1 AND NOT OVEN$IN$TOL(n);
                /* Test for a caution condition */
107  5                 If SETPOINT(n)-TOLERANCE(n) > T$t5(n) OR
                         SETPOINT(n)+TOLERANCE(n) < T$t5(n)
                      then do;
109  6                     STATUS(n)=14;
110  6                     BLOCK1=BLOCK1 OR OVEN$CAUTION(n);
111  6                   end;
112  5                   else BLOCK1=BLOCK1 AND NOT OVEN$CAUTION(n);
                /* Test for a danger condition */
113  5                 If SETPOINT(n)-TOLERANCE(n) > TEMP(n) OR
                         SETPOINT(n)+TOLERANCE(n) < TEMP(n)
                      then do;
115  6                     STATUS(n)=21;
116  6                     BLOCK2=BLOCK2 OR OVEN$DANGER(n);
117  6                   end;
118  5                   else BLOCK2=BLOCK2 AND NOT OVEN$DANGER(n);
                /* Handle control of heater elements */
119  5                 If SETPOINT(n) > T$t10(n)
                      then BLOCK3=BLOCK3 OR OVEN$HEATER(n);
121  5                 else BLOCK3=BLOCK3 AND NOT OVEN$HEATER(n);
122  5               end;
123  4             else do;
                /* Turn everything off when operator shuts off oven */
124  5                 BLOCK1=BLOCK1 AND NOT OVEN$IN$TOL(n);
125  5                 BLOCK1=BLOCK1 AND NOT OVEN$CAUTION(n);
126  5                 BLOCK3=BLOCK3 AND NOT OVEN$HEATER(n);
```

```
127   5                    BLOCK2=BLOCK2 AND NOT OVEN$DANGER(n);
128   5                    BLOCK2=BLOCK2 AND NOT OVEN$RUN(n);
129   5                    STATUS(n)=0;
130   5                end;
131   4                Call RQSEND(.CONSTANT$LOCKOUT$EXCH,MSG$PTR);
132   4            end;

                   /* Output data to real world */
133   3            OUTPUT(232)=BLOCK1;
134   3            OUTPUT(233)=BLOCK2;
135   3            OUTPUT(234)=BLOCK3;
136   3        end;
137   2        end CONTROL$TASK;
138   1        end CONTROL$TASK$MODULE;

                 MODULE INFORMATION:

                      CODE AREA SIZE     = 0946H   2374D
                      VARIABLE AREA SIZE = 0054H     84D
                      MAXIMUM STACK SIZE = 0006H      6D
                      235 LINES READ
                      0 PROGRAM ERROR(S)

                 END OF PL/M-80 COMPILATION


                 $TITLE('CRT PARAMETER TASK')
                 /*********************************************
                 * This task is used to examine and update the *
                 * temperature setpoints and tolerances for    *
                 * each of the four ovens.                     *
                 *********************************************/
   1             UPDATE$TASK:
                 Do;

                 $Include (:F0:COMMON.ELT)
   2   1   =     DECLARE TRUE LITERALLY '0FFH';
   3   1   =     DECLARE FALSE LITERALLY '00H';
   4   1   =     DECLARE BOOLEAN LITERALLY 'BYTE';
   5   1   =     DECLARE FOREVER LITERALLY 'WHILE 1';
                 $Include (:F0:MSGTYP.ELT)
   6   1   =     DECLARE DATA$TYPE LITERALLY '0',
           =             INT$TYPE LITERALLY '1',
           =             MISSED$INT$TYPE LITERALLY '2',
           =             TIME$OUT$TYPE LITERALLY '3',
           =             FS$REQ$TYPE LITERALLY '4',
           =             UC$REQ$TYPE LITERALLY '5',
           =             FS$NAK$TYPE LITERALLY '6',
           =             CNTRL$C$TYPE LITERALLY '7',
           =             READ$TYPE LITERALLY '8',
           =             CLR$RD$TYPE LITERALLY '9',
           =             LAST$RD$TYPE LITERALLY '10',
           =             ALARM$TYPE LITERALLY '11',
           =             WRITE$TYPE LITERALLY '12';
                 $Include (:F0:MSG.ELT)
```

```
 7    1  =    DECLARE MSG$HDR LITERALLY '
          =     LINK ADDRESS,
          =     LENGTH ADDRESS,
          =     TYPE BYTE,
          =     HOME$EX ADDRESS,
          =     RESP$EX ADDRESS';
          =
 8    1  =    DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE(
          =     MSG$HDR,
          =     REMAINDER(1) BYTE)';
               $Include (:F0:THMSG.ELT)
 9    1  =    DECLARE TH$MSG LITERALLY 'STRUCTURE (
          =     MSGHDR,
          =     STATUS ADDRESS,
          =     BUFFER$ADR ADDRESS,
          =     COUNT ADDRESS,
          =     ACTUAL ADDRESS,
          =     REMAINDER(120) BYTE)';
10    1  =    DECLARE MIN$TH$MSG$LENGTH LITERALLY '17';
               $Include (:F0:CHAR.ELT)
          =
          =    /* SPECIAL ASCII CHARACTERS */
          =
11    1  =    DECLARE
          =      NULL             LITERALLY '00H',
          =      CONTROL$C        LITERALLY '03H',
          =      CONTROL$E        LITERALLY '05H',
          =      BELL             LITERALLY '07H',
          =      TAB              LITERALLY '09H',
          =      LF               LITERALLY '0AH',
          =      VT               LITERALLY '0BH',
          =      FF               LITERALLY '0CH',
          =      CR               LITERALLY '0DH',
          =      CONTROL$P        LITERALLY '10H',
          =      CONTROL$Q        LITERALLY '11H',
          =      CONTROL$R        LITERALLY '12H',
          =      CONTROL$S        LITERALLY '13H',
          =      CONTROL$X        LITERALLY '18H',
          =      CONTROL$Z        LITERALLY '1AH',
          =      ESC              LITERALLY '1BH',
          =      QUOTE            LITERALLY '22H',
          =      LCA              LITERALLY '61H',
          =      LCZ              LITERALLY '7AH',
          =      RUBOUT           LITERALLY '7FH';
          =
               $Include (:F0:SYNCH.EXT)
12    1  =    RQSEND:
          =      PROCEDURE (EXCHANGE$POINTER,MESSAGE$POINTER) EXTERNAL;
13    2  =        DECLARE (EXCHANGE$POINTER,MESSAGE$POINTER) ADDRESS;
          =
14    2  =      END RQSEND;
          =
15    1  =    RQWAIT:
          =      PROCEDURE (EXCHANGE$POINTER,DELAY) ADDRESS EXTERNAL;
16    2  =        DECLARE (EXCHANGE$POINTER,DELAY) ADDRESS;
          =
```

```
17   2  =      END RQWAIT;
        =
18   1  =    RQACPT:
        =      PROCEDURE (EXCHANGE$POINTER) ADDRESS EXTERNAL;
19   2  =        DECLARE EXCHANGE$POINTER ADDRESS;
        =
20   2  =      END RQACPT;
        =
21   1  =    RQISND:
        =      PROCEDURE (IED$PTR) EXTERNAL;
22   2  =        DECLARE IED$PTR ADDRESS;
        =
23   2  =      END RQISND;
24   1        Declare TEMP$CALIBRATE(5) address external;
25   1        Declare UPDATE$EXCH(5) address external;
26   1        Declare CRT$STATUS$EXCH(5) address external;
27   1        Declare COMP$EXCH(5) address external;
28   1        Declare CONSTANT$LOCKOUT$EXCH(5) address external;
29   1        Declare RQOUTX(5) address external;
30   1        Declare RQINPX(5) address external;
31   1        Declare WORDS$EXCH(5) address external;
32   1        Declare SETPOINT(4) address external;
33   1        Declare TOLERANCE(4) address external;
34   1        Declare BUFFER2 address;
35   1        Declare MSG$PTR address;
36   1        Declare MSG structure (
                 MSG$HDR,
                 STATUS address,
                 BUFFER$PTR address,
                 COUNT address,
                 ACTUAL address );
37   1        Declare CAL$TEMP structure (
                 MSG$HDR,
                 CAL address );
38   1        Declare UPD$MSG address;
39   1        Declare ENERGIZE based UPD$MSG structure (
                 MSG$HDR,
                 STATUS address,
                 BUFFER$PTR address,
                 COUNT address,
                 ACTUAL address );
40   1        Declare ENABLE$MSG structure (
                 MSG$HDR );
41   1        Declare BUFFER(80) byte;
42   1        Declare OVEN byte;
43   1          DEC$REP:
                 Procedure (SOURCE,TARGET) external;
44   2            Declare (SOURCE,TARGET) address;
45   2        end DEC$REP;
46   1          ASC$2$BINARY:
                    Procedure (SOURCE,TARGET,SIZE) byte external;
47   2            Declare (SOURCE,TARGET) address;
48   2            Declare SIZE byte;
49   2        end ASC$2$BINARY;
50   1          Declare MSG$1(20) byte data (
                   ESC,'E','ENTER OVEN NUMBER-');
```

```
51    1       Declare MSG$2(28) byte data (
                  CR,LF,
                  'ENTER NEW SETPOINT-',
                  'XXXX.X-' );
52    1       Declare MSG$3(29) byte data (
                  CR,LF,
                  'ENTER NEW TOLERANCE-',
                  'XXXX.X-' );
53    1       Declare CALMSG(12) byte data (
                  'TEMPERATURE-' );
54    1       Declare MSG$4(62) byte data (
                  CR,LF,
                  '(STATUS-(S), PARAMETERS-(P), CALIBRATE-(C))',
                  CR,LF,
                  'ENTER REQUEST-' );
55    1       Declare WAIT literally 'MSG$PTR=';
56    1       Declare FOR literally 'RQWAIT';
57    1       Declare START literally 'CALL';
58    1       Declare TASK literally 'RQSEND';
59    1     UPDATE:
        Procedure public;
          /* Initialize task at start-up time */
60    2       Do forever;
61    3       MSG.RESP$EX=.COMP$EXCH;
          /* Wait for request to enter task */
62    3            UPD$MSG=RQWAIT (.UPDATE$EXCH,0);
          /* Get desired oven number from operator */
63    3            RQST$OVEN:
                   MSG.BUFFER$PTR=.MSG$1;
64    3            MSG.TYPE=WRITE$TYPE;
65    3            MSG.COUNT=20;
66    3            Start task (.RQOUTX,.MSG);
67    3            Wait for (.COMP$EXCH,0);
          /* ...Input new number */
68    3            MSG.BUFFER$PTR=.BUFFER;
69    3            MSG.COUNT=255;
70    3            MSG.TYPE=CLR$RD$TYPE;
71    3            Start task (.RQINPX,.MSG);
72    3            Wait for (.COMP$EXCH,0);
73    3            OVEN=(BUFFER(0) AND 07E)-1;
74    3            If OVEN >3 then go$to RQST$OVEN;
          /* Display request and current setpoint */
76    3            GET$TEMP:
                   Call move (28,.MSG$2,.BUFFER);
77    3            Call DEC$REP (.SETPOINT(oven),.BUFFER+21);
78    3              MSG.TYPE=WRITE$TYPE;
79    3            MSG.COUNT=28;
80    3            Start task (.RQOUTX,.MSG);
81    3            Wait for (.COMP$EXCH,0);
          /* ... Input new setpoint */
82    3            MSG.TYPE=CLR$RD$TYPE;
83    3            Start task (.RQINPX,.MSG);
84    3            Wait for (.COMP$EXCH,0);
85    3            If ASC$2$BINARY(.BUFFER,.BUFFER2,1)=0 OR BUFFER2 > 700
                   then go$to GET$TEMP;
87    3            If BUFFER2 <> 0
                   then do;
89    4                Wait for (.CONSTANT$LOCKOUT$EXCH,0);
90    4                SETPOINT(oven)=BUFFER2;
91    4                Start task (.CONSTANT$LOCKOUT$EXCH,MSG$PTR);
```

```
 92   4          end;
           /* Display request and current tolerance */
 93   3          GET$TOL:
                 Call move (29,.MSG$3,.BUFFER);
 94   3          Call DEC$REP (.TOLERANCE(oven),.BUFFER+22);
 95   3          MSG.TYPE=WRITE$TYPE;
 96   3          MSG.COUNT=29;
 97   3          Start task (.RQOUTX,.MSG);
 98   3          Wait for (.COMP$EXCH,0);
           /* ...Input new tolerance */
 99   3          MSG.TYPE=CLR$RD$TYPE;
100   3          Start task (.RQINPX,.MSG);
101   3          Wait for (.COMP$EXCH,0);
102   3          If ASC$2$BINARY(.BUFFER,.BUFFER2,1)=0 OR BUFFER2 > 700
                 then go$to GET$TOL;
104   3          If BUFFER2 <> 0
                 then do;
106   4              Wait for (.CONSTANT$LOCKOUT$EXCH,0);
107   4              TOLERANCE(oven)=BUFFER2;
108   4              Start task (.CONSTANT$LOCKOUT$EXCH,MSG$PTR);
109   4          end;
           /* Ask operator if he is finished */
110   3          REQ$NEXT:
                 MSG.TYPE=WRITE$TYPE;
111   3          MSG.COUNT=52;
112   3          MSG.BUFFER$PTR=.MSG$4;
113   3          Start task (.RQOUTX,.MSG);
114   3          Wait for (.COMP$EXCH,0);
           /* ...Get his response */
115   3          MSG.TYPE=CLR$RD$TYPE;
116   3          MSG.BUFFER$PTR=.BUFFER;
117   3          Start task (.RQINPX,.MSG);
118   3          Wait for (.COMP$EXCH,0);
119   3          If (BUFFER(0) <>'S' AND BUFFER(0) <>'P'
                  AND BUFFER(0) <> 'C')
                 then go$to REQ$NEXT;
121   3          If BUFFER(0)='P'
                   then go$to RQST$OVEN;
123   3          If BUFFER(0)='C'
                 then do;
125   4              GET$CAL:
                     MSG.TYPE=WRITE$TYPE;
126   4              MSG.COUNT=12;
127   4              MSG.BUFFER$PTR=.CALMSG;
128   4              Start task (.RQOUTX,.MSG);
129   4              Wait for (.COMP$EXCH,0);
130   4              MSG.TYPE=CLR$RD$TYPE;
131   4              MSG.BUFFER$PTR=.BUFFER;
132   4              Start task (.RQINPX,.MSG);
133   4              Wait for (.COMP$EXCH,0);
134   4              If ASC$2$BINARY(.BUFFER,.BUFFER2,1) =0
                       OR BUFFER2>350 OR BUFFER2<200
                     then go$to GET$CAL;
136   4              CAL$TEMP.CAL=BUFFER2;
137   4              Call RQSEND (.TEMP$CALIBRATE,.CAL$TEMP);
138   4          end;
```

```
                MODULE INFORMATION:
                     CODE AREA SIZE      = 03C3H     963D
                     VARIABLE AREA SIZE  = 007CH     124D
                     MAXIMUM STACK SIZE  = 0004H       4D
                     264 LINES READ
                     0 PROGRAM ERROR(S)
                END OF PL/M-80 COMPILATION

139    3          ENERGIZE.TYPE=100;
140    3          Start task (.CRT$STATUS$EXCH,UPD$MSG);

141    3       end;
142    2    end UPDATE;
143    1 end UPDATE$TASK;



                $TITLE('CRT UPDATE TASK')
                /************************************************
                * This task is utilized to update the CRT ter-  *
                * minal display with the current operating par- *
                * ameters. It will be entered upon sytem start- *
                * up, upon operator request, or when a problem  *
                * exists with any of the activated ovens.       *
                ************************************************/
 1              CRT$DATA$MODULE:
                Do;
                $INCLUDE(:F0:SYNCH.EXT)
 2    1    =    RQSEND:
      =         PROCEDURE (EXCHANGE$POINTER,MESSAGE$POINTER) EXTERNAL;
 3    2    =      DECLARE (EXCHANGE$POINTER,MESSAGE$POINTER) ADDRESS;
      =
 4    2    =      END RQSEND;
      =
 5    1    =    RQWAIT:
      =         PROCEDURE (EXCHANGE$POINTER,DELAY) ADDRESS EXTERNAL;
 6    2    =      DECLARE (EXCHANGE$POINTER,DELAY) ADDRESS;
      =
 7    2    =      END RQWAIT;
      =
 8    1    =    RQACPT:
      =         PROCEDURE (EXCHANGE$POINTER) ADDRESS EXTERNAL;
 9    2    =      DECLARE EXCHANGE$POINTER ADDRESS;
      =
10    2    =      END RQACPT;
      =
11    1    =    RQISND:
      =         PROCEDURE (IED$PTR) EXTERNAL;
12    2    =      DECLARE IED$PTR ADDRESS;
      =
13    2    =      END RQISND;
                $INCLUDE (:F0:MSGTYP.ELT)
14    1    =    DECLARE DATA$TYPE LITERALLY '0',
```

```
          =           INT$TYPE LITERALLY '1',
          =           MISSED$INT$TYPE LITERALLY '2',
          =           TIME$OUT$TYPE LITERALLY '3',
          =           FS$REQ$TYPE LITERALLY '4',
          =           UC$REQ$TYPE LITERALLY '5',
          =           FS$NAK$TYPE LITERALLY '6',
          =           CNTRL$C$TYPE LITERALLY '7',
          =           READ$TYPE LITERALLY '8',
          =           CLR$RD$TYPE LITERALLY '9',
          =           LAST$RD$TYPE LITERALLY '10',
          =           ALARM$TYPE LITERALLY '11',
          =           WRITE$TYPE LITERALLY '12';
              $INCLUDE (:F0:EXCH.ELT)
15    1   =   DECLARE EXCHANGE$DESCRIPTOR LITERALLY 'STRUCTURE (
          =    MESSAGE$HEAD ADDRESS,
          =    MESSAGE$TAIL ADDRESS,
          =    TASK$HEAD ADDRESS,
          =    TASK$TAIL ADDRESS,
          =    EXCHANGE$LINK ADDRESS)';
              $INCLUDE (:F0:COMMON.ELT)
16    1   =   DECLARE TRUE LITERALLY '0FFH';
17    1   =   DECLARE FALSE LITERALLY '00H';
18    1   =   DECLARE BOOLEAN LITERALLY 'BYTE';
19    1   =   DECLARE FOREVER LITERALLY 'WHILE 1';
              $INCLUDE (:F0:MSG.ELT)
20    1   =   DECLARE MSG$HDR LITERALLY '
          =    LINK ADDRESS,
          =    LENGTH ADDRESS,
          =    TYPE BYTE,
          =    HOME$EX ADDRESS,
          =    RESP$EX ADDRESS';
          =
21    1   =   DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE(
          =    MSG$HDR,
          =    REMAINDER(1) BYTE)';
              $INCLUDE (:F0:CHAR.ELT)
          =
          =   /* SPECIAL ASCII CHARACTERS */
          =
22    1   =   DECLARE
          =     NULL            LITERALLY '00H',
          =     CONTROL$C       LITERALLY '03H',
          =     CONTROL$E       LITERALLY '05H',
          =     BELL            LITERALLY '07H',
          =     TAB             LITERALLY '09H',
          =     LF              LITERALLY '0AH',
          =     VT              LITERALLY '0BH',
          =     FF              LITERALLY '0CH',
          =     CR              LITERALLY '0DH',
          =     CONTROL$P       LITERALLY '10H',
          =     CONTROL$Q       LITERALLY '11H',
          =     CONTROL$R       LITERALLY '12H',
          =     CONTROL$S       LITERALLY '13H',
          =     CONTROL$X       LITERALLY '18H',
          =     CONTROL$Z       LITERALLY '1AH',
          =     ESC             LITERALLY '1BH',
          =     QUOTE           LITERALLY '22H',
```

```
          =     LCA             LITERALLY '61H',
          =     LCZ             LITERALLY '7AH',
          =     RUBOUT          LITERALLY '7FH';
          =
              $INCLUDE (:F?:THMSG.ELT)
23    1   =   DECLARE TH$MSG LITERALLY 'STRUCTURE (
          =    MSGHDR,
          =    STATUS ADDRESS,
          =    BUFFER$ADR ADDRESS,
          =    COUNT ADDRESS,
          =    ACTUAL ADDRESS,
          =    REMAINDER(128) BYTE)';
24    1   =   DECLARE MIN$TH$MSG$LENGTH LITERALLY '17';
25    1       Declare HOME literally '1BH,48H';
26    1       Declare L1$IMAGE(90) byte data (
                Home,Lf,Lf,Lf,Lf,Lf,
                'TEMPERATURE      ',
                '                 ',
                '                 ',
                '                 ',
                '                   ',
                'DEGREES C.' );
27    1     Declare L2$IMAGE(92) byte data (
                Home,Lf,Lf,Lf,Lf,Lf,Lf,Lf,
                'SETPOINT         ',
                '                 ',
                '                 ',
                '                 ',
                '                   ',
                'DEGREES C.' );
28    1     Declare L3$IMAGE(94) byte data (
                Home,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,
                'TOLERANCE        ',
                '                 ',
                '                 ',
                '                 ',
                '                   ',
                'DEGREES C.' );
29    1     Declare L4$IMAGE(75) byte data (
                Home,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,
                'STATUS           ',
                '  OFF            ',
                '  OFF            ',
                '  OFF            ',
                '  OFF   ' );
30    1     Declare CRT$HDR(168) byte data (
                1BH,45H,'                                        ',
                'OVEN STATUS DISPLAY',
                Cr,Lf,Lf,'                    ',
                  'OVEN-1         ',
                'OVEN-2           ',
                'OVEN-3           ',
                'OVEN-4',
```

```
                Cr,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,Lf,
            Lf,
                'TYPE ESCAPE TO ADJUST SETPOINTS' );
31    1     Declare BELLS(4) byte data (
                Bell,Bell,Bell,Bell );
32    1     Declare MESSAGES(35) byte data (
                '  OFF  ',
                '   OK  ',
                'CAUTION',
                ' ALARM ',
                '       ' );
33    1       Declare DISPLAY$PTR1(4) address data (
                .WORK$BUFF+23,
                .WORK$BUFF+36,
                .WORK$BUFF+49,
                .WORK$BUFF+62 );
34    1     Declare DISPLAY$PTR2(4) address data (
                .WORK$BUFF+25,
                .WORK$BUFF+38,
                .WORK$BUFF+51,
                .WORK$BUFF+64 );
35    1     Declare DISPLAY$PTR3(4) address data (
                .WORK$BUFF+27,
                .WORK$BUFF+40,
                .WORK$BUFF+53,
                .WORK$BUFF+66 );
36    1     Declare DISPLAY$PTR4(4) address data (
                .WORKBUFF+30,
                .WORKBUFF+43,
                .WORKBUFF+56,
                .WORKBUFF+69 );
37    1     Declare MSG$PTR address;
38    1     Declare MSG based MSG$PTR structure (
                MSG$HDR,
                COUNT address );
39    1     Declare STARTER(3) structure (
                MSG$HDR );
40    1     Declare READ structure (
                MSG$HDR,
                STATUS address,
                BUFFER$PTR address,
                COUNT address,
                ACTUAL address );
41    1     Declare DISPLAY$TEMP(4) structure (
                UPPER address,
                LOWER address );
42    1     Declare DISPLAY$SET(4) structure (
                LOWER address,
                UPPER address );
43    1     Declare DISPLAY$TOL(4) structure (
                LOWER address,
                UPPER address );
```

```
44   1          Declare OVEN$ON(4) byte data (
                   01H,02H,04H,08H );
45   1        Declare OVEN$CAUTION(4) byte data (
                   10H,20H,40H,80H );
46   1          Declare CRT structure (
                   MSG$HDR,
                   STATUS address,
                   BUFFER$PTR address,
                   COUNT address,
                   ACTUAL address );
47   1          Declare CRTLOCK structure (MSG$HDR);
48   1          Declare CRT$DISPLAY$LOCK(5) address external;
49   1        Declare TEMP$LOCKOUT$EXCH(5) address external;
50   1        Declare CONSTANT$LOCKOUT$EXCH(5) address external;
51   1        Declare CRT$EXCH(5) address external;
52   1        Declare CRT$STATUS$EXCH(5) address external;
53   1        Declare DUMMY$EXCH(5) address external;
54   1        Declare READ$BUFFER$EXCH(5) address external;
55   1        Declare UPDATE$EXCH(5) address external;
56   1        Declare RQINPX(5) address external;
57   1        Declare RQOUTX(5) address external;
58   1        Declare RQWAKE(5) address external;
59   1        Declare RQL7EX(5) address external;
60   1        Declare RQL6EX(5) address external;
61   1        Declare RQDBUC(5) address external;
62   1        Declare RQALRM(5) address external;
63   1          Declare TEMP(4) address external;
64   1        Declare DISP$TEMP(4) address;
65   1        Declare SETPOINT(4) address external;
66   1        Declare DISP$SETPNT(4) address;
67   1        Declare TOLERANCE(4) address external;
68   1        Declare DISP$TOL(4) address;
69   1        Declare STATUS(4) byte external;
70   1        Declare DISP$STAT(4) byte;
71   1        Declare (BLOCK1,BLOCK2) byte external;
72   1        Declare WORK$BUFF(170) byte;
73   1        Declare BUFFER$A(70) byte;
74   1        Declare (CHANGE,n,ALARM,NEW,BLANKER) byte;
75   1          Declare START literally 'call';
76   1        Declare TASK literally 'rqsend';
77   1        Declare WAIT literally 'msg$ptr=';
78   1        Declare For literally 'rqwait';
79   1          DEC$REP:
                   Procedure(SOURCE,TARGET) external;
80   2          Declare (SOURCE,TARGET) address;
81   2        end DEC$REP;
```

```
 82   1      CRT$DATA$TASK:
             Procedure public;
             /* Initialize system at start-up time */
 83   2        Start task (.TEMP$LOCKOUT$EXCH,.STARTER(0));
 84   2        Start task (.CONSTANT$LOCKOUT$EXCH,.STARTER(1));
 85   2        STARTER(2).TYPE=100;
 86   2        Start task (.CRT$STATUS$EXCH,.STARTER(2));
 87   2        CRT.RESP$EX=.CRT$EXCH;
             /* Perform main CRT wait */
 88   2        Do forever;
 89   3            Wait for (.DUMMY$EXCH,10);
 90   3            Wait for (.CRT$STATUS$EXCH,0);
 91   3            If MSG.TYPE=255
                  then ALARM=1;
 93   3            else ALARM=0;
             /* Output heading */
 94   3            If (MSG.TYPE=100 OR MSG.TYPE=255)
                  then do;
 96   4                If ALARM=0
                        then call RQSEND(.CRT$DISPLAY$LOCK,.CRTLOCK);
 98   4                CRT.TYPE=WRITE$TYPE;
 99   4                CRT.COUNT=167;
100   4                CRT.BUFFER$PTR=.WORK$BUFF;
101   4                READ.TYPE=CLR$RD$TYPE;
102   4                READ.COUNT=255;
103   4                READ.RESP$EX=.READ$BUFFER$EXCH;
104   4                READ.BUFFER$PTR=.BUFFERA;
105   4                If ALARM=0
                       then start task (.RQINPX,.READ);
107   4                Call move (82,.CRT$HDR,.WORK$BUFF);
108   4                Call move (86,.CRT$HDR+82,.WORK$BUFF+82);
109   4                Start task (.RQOUTX,.CRT);
110   4                Wait for (.CRT$EXCH,0);
111   4                NEW=1;
112   4            end;
             /* Test for change in temperature of any oven */
113   3            CHANGE=0;
114   3            Wait for (.TEMP$LOCKOUT$EXCH,0);
115   3            Do n=0 to 3;
116   4                If TEMP(n)<>DISP$TEMP(n)
                     then CHANGE=1;
118   4            end;
119   3            Call move (8,.TEMP,.DISP$TEMP);
120   3            Start task (.TEMP$LOCKOUT$EXCH,MSG$PTR);
             /* When a change exists build new line */
121   3            If CHANGE OR NEW
                  then do;
123   4                Call move (90,.L1$IMAGE,.WORK$BUFF);
124   4                Do n=0 to 3;
125   5                    Call DEC$REP(.DISP$TEMP(n),DISPLAY$PTR1(n));
126   5                end;
             /* Output new temperature line to CRT */
127   4                  CRT.TYPE=WRITE$TYPE;
128   4                CRT.COUNT=87;
```

```
129   4             Start task (.RQOUTX,.CRT);
130   4             Wait for (.CRT$EXCH,0);
131   4         end;
          /* Test for change in oven setpoints */
132   3         CHANGE=0;
133   3         Wait for (.CONSTANT$LOCKOUT$EXCH,0);
134   3         Do n=0 to 3;
135   4             If SETPOINT(n)<>DISP$SETPNT(n)
                  then CHANGE=1;
137   4         end;
138   3         Call move (8,.SETPOINT,.DISP$SETPNT);
139   3         Start task (.CONSTANT$LOCKOUT$EXCH,MSG$PTR);
          /* Build new line when a change was detected */
140   3         If CHANGE OR NEW
                  then do;
142   4             Call move (92,.L2$IMAGE,.WORKBUFF);
143   4             Do n=0 to 3;
144   5                 Call DEC$REP(.DISP$SETPNT(n),DISPLAY$PTR2(n));
145   5             end;
          /* Output setpoint line */
146   4             CRT.TYPE=WRITE$TYPE;
147   4             CRT.COUNT=89;
148   4             CRT.BUFFER$PTR=.WORKBUFF;
149   4             Start task (.RQOUTX,.CRT);
150   4             Wait for (.CRT$EXCH,0);
151   4         end;
          /* Test for change in tolerance line */
152   3         CHANGE=0;
153   3         Wait for (.CONSTANT$LOCKOUT$EXCH,0);
154   3         Do n=0 to 3;
155   4             If TOLERANCE(n)<>DISP$TOL(n)
                  then CHANGE=1;
157   4         end;
158   3         Call move (8,.TOLERANCE,.DISP$TOL);
159   3         Start task (.CONSTANT$LOCKOUT$EXCH,MSG$PTR);
          /* When change is found, build new line */
160   3         If CHANGE OR NEW
                  then do;
162   4             Call move (94,.L3$IMAGE,.WORK$BUFF);
163   4             Do n=0 to 3;
164   5                 Call DEC$REP(.DISP$TOL(n),DISPLAY$PTR3(n));
165   5             end;
          /* Output tolerance line */
166   4             CRT.TYPE=WRITE$TYPE;
167   4             CRT.COUNT=91;
168   4             CRT.BUFFER$PTR=.WORKBUFF;
169   4             Start task (.RQOUTX,.CRT);
170   4             Wait for (.CRT$EXCH,0);
171   4         end;
            /* Build status message */
172   3         CHANGE=0;
173   3         Wait for (.CONSTANT$LOCKOUT$EXCH,0);
174   3         Do n=0 to 3;
175   4             If STATUS(n)<>DISP$STAT(n)
                  then CHANGE=1;
```

```
177   4           end;
178   3           Call move (4,.STATUS,.DISP$STAT);
179   3           Start task (.CONSTANT$LOCKOUT$EXCH,MSG$PTR);
          /* Output to display */
180   3           If CHANGE OR NEW
                  then do;
182   4               Call move (75,.L4IMAGE,.WORK$BUFF);
183   4               Do n=0 to 3;
184   5                   Call move(7,.MESSAGES+DISP$STAT(n),DISPLAY$PTR4(
          n));
185   5               end;
186   4               CRT.COUNT=76;
187   4               Start task (.RQOUTX,.CRT);
188   4               Wait for (.CRT$EXCH,0);
189   4           end;
          /* test for request to exit this mode */
190   3           MSG$PTR=RQACPT (.READ$BUFFER$EXCH);
191   3           If ALARM=0
                  then do;
193   4               If (MSG$PTR <> 0 and BUFFERA(0) = 1BH)
                      then do;
195   5                   MSG$PTR=RQWAIT(.CRT$DISPLAY$LOCK,0);
196   5                   start task (.UPDATE$EXCH,MSG$PTR);
197   5               end;
198   4               else do;
199   5                   If MSG$PTR=0
                          then STARTER(2).TYPE=200;
201   5                   else STARTER(2).TYPE=100;
202   5                   Start task (.CRT$STATUS$EXCH,.STARTER(2));
203   5                   NEW=0;
204   5               end;
205   4           end;
206   3       end;
207   2   end CRT$DATA$TASK;
208   1 end CRT$DATA$MODULE;


MODULE INFORMATION:
     CODE AREA SIZE     = 0720H    1824D
     VARIABLE AREA SIZE = 0189H     393D
     MAXIMUM STACK SIZE = 0004H       4D
     388 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
        $TITLE('ASCII STRING TO FIXED BINARY')
        /*******************************************************
        * This program converts an ASCII string into a fix- *
        * ed point binary number.  The fixed decimal point  *
        * is determined by the parameter passed in SIZE.    *
        *******************************************************/
1       ASC$2$BINARY$MODULE:
        Do;
        /* SPECIAL ASCII CHARACTERS */
2   1   DECLARE
          NULL          LITERALLY '00H',
          CONTROL$C     LITERALLY '03H',
          CONTROL$E     LITERALLY '05H',
          BELL          LITERALLY '07H',
          TAB           LITERALLY '09H',
          LF            LITERALLY '0AH',
          VT            LITERALLY '0BH',
          FF            LITERALLY '0CH',
          CR            LITERALLY '0DH',
          CONTROL$P     LITERALLY '10H',
          CONTROL$Q     LITERALLY '11H',
          CONTROL$R     LITERALLY '12H',
          CONTROL$S     LITERALLY '13H',
          CONTROL$X     LITERALLY '18H',
          CONTROL$Z     LITERALLY '1AH',
          ESC           LITERALLY '1BH',
          QUOTE         LITERALLY '22H',
          LCA           LITERALLY '61H',
          LCZ           LITERALLY '7AH',
          RUBOUT        LITERALLY '7FH';

3   1   ASC$2$BINARY:
        Procedure (SRC$PTR,TRGT$PTR,SIZE) byte public;
4   2       Declare (SRC$PTR,TRGT$PTR) address;
5   2       Declare (SOURCE based SRC$PTR)(80) byte;
6   2       Declare RESULT based TRGT$PTR address;
7   2       Declare (N,SIZE,K,DP,DIGITS,VALID) byte;
8   2       Declare POWER(6) address data (
                0,1,10,100,1000,10000 );
          /* Find location of decimal point */
9   2         n=0;
10  2         Do while SOURCE(n)<>'.' AND SOURCE(n)<>CR
                    AND SOURCE(n)<>LF;
11  3             n=n+1;
12  3           end;
13  2         DP=n;
          /* Provide correct number of digits to right of decimal */
14  2         Do n=0 to SIZE;
15  3             SOURCE(DP+n)=SOURCE(DP+n+1);
16  3             If SOURCE(DP+n)>39H OR SOURCE(DP+n)<30H
                  then do k=n to SIZE;
18  4                 SOURCE(DP+k)='0';
19  4             end;
```

```
20    3           end;
                /* Mark end of string */
21    2           DIGITS=DP+SIZE;
                /* Test for all valid characters */
22    2           VALID=1;
23    2           Do n=0 to DIGITS;
24    3               If SOURCE(n)>39H OR SOURCE(n)<30H
                        then VALID=0;
26    3           end;
27    2           If DIGITS>5
                    then VALID=0;
                /* Convert data to binary and store */
29    2           n=0;
30    2           If VALID=1
                    then do;
32    3               RESULT=0;
33    3               Do while DIGITS > 0;
34    4                   RESULT=RESULT+((
                                SOURCE(n) AND 0FH) * POWER(DIGITS));
35    4                   n=n+1;
36    4                   DIGITS=DIGITS-1;
37    4               end;
38    3           end;
                /* Return to calling program */
39    2           Return VALID;
40    2         end ASC$2$BINARY;
41    1       end ASC$2$BINARY$MODULE;


MODULE INFORMATION:

     CODE AREA SIZE      = 0178H     376D
     VARIABLE AREA SIZE  = 00CAH     10CD
     MAXIMUM STACK SIZE  = 0004H       4D
     80 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
                $TITLE('COMMON VARIABLE STORAGE')
                /*************************************************
                * This module contains those variables common to  *
                * multiple tasks in the oven control application. *
                *************************************************/
  1             VARIABLE$STORAGE:
                Do;
  2     1       Declare SETPOINT(4) address public;
  3     1       Declare TOLERANCE(4) address public;
  4     1       Declare TEMP(4) address public;
  5     1       Declare STATUS(4) byte public;
  6     1       Declare BLOCK0 byte public;
  7     1       Declare BLOCK1 byte public;
  8     1       Declare BLOCK2 byte public;
  9     1       Declare BLOCK3 byte public;
 10     1       end VARIABLE$STORAGE;


MODULE INFORMATION:

     CODE AREA SIZE     = 0000H      0D
     VARIABLE AREA SIZE = 0020H     32D
     MAXIMUM STACK SIZE = 0000H      0D
     16 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
                $TITLE('WORD TO ASCII CONVERSION')
                /*******************************************************
                * This routine converts a fixed point word in mem-     *
                * ory into a 4 digit plus 1 decimal ASCII display-     *
                * able number.  Zero blanking is included.             *
                *******************************************************/
  1             DEC$REP$MODULE:
                Do;

  2   1         DEC$REP:
                Procedure (SOURCE,TARGET) public ;
  3   2           Declare (SOURCE,TARGET) address;
  4   2           Declare ANSWR(5) byte;
  5   2           Declare (DISPLAY based TARGET)(5) byte;
  6   2           Declare NUMBER based SOURCE structure (
                      ELEMENT address );
  7   2           Declare N byte;
  8   2           Declare CALC(5) address;
                /* Initialize */
  9   2           Do n=0 to 4;
 10   3               ANSWR(n)='0';
 11   3           end;
 12   2           CALC(0)=NUMBER.ELEMENT;
                /* Convert to ASCII */
 13   2           Do n=1 to 5;
 14   3               CALC(n)=CALC(n-1)/10;
 15   3               ANSWR(5-n)=(CALC(n-1) mod 10) + 30H;
 16   3           end;
                /* Perform zero blanking */
 17   2           Do n=0 to 3;
 18   3               If ANSWR(n)<>'0'
                      then n=4;
 20   3               else ANSWR(n)=' ';
 21   3           end;
                /* Format with decimal point */
 22   2           Call move (4,.ANSWR,TARGET);
 23   2           DISPLAY(4)='.';
 24   2           DISPLAY(5)=ANSWR(4);
 25   2         end DEC$REP;
 26   1       end DEC$REP$MODULE;


MODULE INFORMATION:

     CODE AREA SIZE      = 00EEH     238D
     VARIABLE AREA SIZE  = 0014H      20D
     MAXIMUM STACK SIZE  = 0004H       4D
     40 LINES READ
     0 PROGRAM ERROR(S)
END OF PL/M-80 COMPILATION
```